ネットワーク分散型並列処理システムを実現する通信機構

A Communication Mechanism for Parallel Processing on a Distributed System

伊達 新哉

平成12年2月9日

内容梗概

ネットワークで接続された分散システムは、システム全体として莫大な計算資源を潜在している。それらを並列処理によって有効利用し、理論上同等性能を持つ並列計算機に比肩する性能を実現するためには、通信性能や信頼性などの分散システムにおける通信の問題を解決する通信機構が必要である。

本稿では、LAN程度の規模の分散システムにおいて比較的細粒度の並列処理を実用レベルにまで高めることを目指す通信機構について述べる。本通信機構は上記の問題を解決するため、通信最適化のサポート, ユーザレベル通信の実現, 信頼性保証, デッドロック防止の4つの機能を実現する。

システムバスに接続したネットワーク・インタフェースが主記憶に対するキャッシュタグを保持し、システムバスのコヒーレンスプロトコルを利用する。このことにより主記憶を大容量のキャッシュとして利用できる環境を提供し、リモートメモリのキャッシングによる通信最適化のサポートを実現する。

ネットワーク・インタフェースは、物理アドレスによって通信を関知する。そのため、通信時のシステムコールが不要になり、オペレーティング・システムの関与しないユーザレベル通信を実現する。また主記憶の一部をネットワーク・インタフェース上にキャッシングし、通信時の処理を一部高速化することが可能である。

従来の上位プロトコルによる処理に代わり、データリンクレベルでの信頼性保証処理として、パケットの順序管理, 送信パケットの管理, 再送処理, フロー制御を主にハードウェアで行ない、プロトコルオーバヘッドを削減する。

ネットワーク・インタフェースにおいてバッファの多重化とプロトコル処理に利用する再送バッファを流用することにより、デッドロック防止を実現する。

製作したプロトタイプ・ハードウェアのネットワーク・インタフェースにおいて、リモートメモリアクセスを行なう場合の通信処理時間の見積りを行なった。その結果、理想的な状態で234サイクル程度でアクセス可能であり、本通信機構が細粒度並列処理に必要なμs単位のレイテンシを実現するために有効であることを確認した。

Abstract

Distributed systems have a large amount of computation power potentially. A communication mechanism which enables high performance computing such as Massively Parallel Processors for distributed systems must solve problems on distributed environment.

In this thesis we propose a communication mechanism for fine-grain parallel processing on a local area distributed system. Specific functions of this mechanism are a remote memory caching mechanism, an user-level communication method, a communication protocol for network reliability, and deadlock prevention.

The network interface of the mechanism, which connected to a system bus, uses main memory as a large size of cache by maintaining cache coherency with main memory tags, and optimizes remote memory accesses.

The network interface snoops a physical address on a system bus and starts a user-level communication without operating system intervention. It supports high speed communications by caching a part of main memory.

To achieve network reliability and reducing protocol overheads, in a data-link-level communication protocol of the mechanism, hardware supports packet sequential management, re-send packet management, retry, and flow control mechanism.

To prevent deadlock, the network interface has separate request and reply queues on each send and receive port, and it uses re-send buffer.

We estimated the latency of remote memory access on prototype hardware. According to the estimation, the latency is approximately 234 cycles in the ideal state, and the communication mechanism is effective for the fine-grain parallel processing, which requires that the latency is approximately a few microseconds.

目 次

1 はじめに 2

2 Computer Colony 3
  2.1 Computer Colony の目標 3
  2.2 実現のための技術 4
  2.3 システムモデル 4
    2.3.1 構成 4
    2.3.2 動作モデル 5

3 ネットワーク分散型並列処理システムを実現する通信機構 6
  3.1 分散環境における通信の問題 6
  3.2 種々の通信機構 6
    3.2.1 メッセージベースの通信機構 . . 6
    3.2.2 分散共有メモリベースの通信機構 7
  3.3 本通信機構のアプローチ 8

4 通信機構の設計 9
  4.1 キャッシングによる通信最適化のサポート 9
    4.1.1 分散共有メモリの構成 9
    4.1.2 システムバスを利用した通信 . . 10
    4.1.3 主記憶タグによるコヒーレンス制御 11
  4.2 ユーザレベル通信の実現 11
    4.2.1 通信資源のプロテクション 12
    4.2.2 自律的な通信の起動 12
    4.2.3 ネットワークアドレス変換 12
    4.2.4 通信先ノードの特定 13
  4.3 軽量通信プロトコルによる信頼性保証 . . 13
    4.3.1 パケットの順序管理 14
    4.3.2 送信済パケットの管理 15
    4.3.3 再送処理 15
    4.3.4 フロー制御 16
    4.3.5 軽量通信プロトコルのまとめ . . 17
    4.3.6 ピギーバック 17
  4.4 デッドロック対策 18
    4.4.1 送受信バッファの多重化 18
    4.4.2 再送バッファの流用 19

5 通信機構の実装と予備的評価 19
  5.1 プロトタイプ・ハードウェア 19
    5.1.1 プロトタイプ・ハードウェアの構成 19
  5.2 ネットワーク・インタフェースの構成 . . 20
  5.3 通信処理時間の見積り 22
    5.3.1 トランザクションとパケット . . 22
    5.3.2 通信処理のフェーズ 22
    5.3.3 各フェーズ処理時間の見積り . . 22
    5.3.4 レイテンシについての考察 24

6 おわりに 26

1 はじめに

現在の計算機システムには2つの流れがある。ピーク性能の追求を目指すスーパーコンピューティング分野と、仕事の能率, 快適性を目指す分散コンピューティング分野である。近年、マイクロプロセッサの単体性能の劇的な向上、ネットワークの高速化により、その両者の融合ともいうべき分野が新たに発生してきている。すなわち、スーパーコンピューティング分野では、従来からのピーク性能の追求のみを目指す分野とは別に、より価格性能比のよいシステムを目指して、ベクトル・スーパーコンピュータからMPP(Massively Parallel Processor)、クラスタという分散化の流れが進みつつある。一方、分散コンピューティング分野では、ペーパーレスオフィスの概念から個人使用, 共用資源,LAN(Local Area Nework)、そしてインターネットワーキングというスーパーコンピューティング分野とは異なる一連の流れが進んでいる。分散コンピューティング分野は、究極的にはネットワークを一つのコンピュータにする、という目標に向かうものである。しかし、そのような流れとは別にネットワークで結合された計算機資源を有効利用し、さらに高性能なシステムとしての実現を目指す研究も盛んになってきた。これらの流れは、高性能と快適性を両立する分散システ

ムにおけるスーパーコンピューティングを目指すものであるといえよう。

しかし、スーパーコンピューティング分野からのアプローチとしての種々のクラスタシステムにおいては、クラスタリングによる分散化のメリットは、ほとんど価格性能比のよさだけに留まっている。しかもスーパーコンピュータの代替品として登場したクラスタの利用形態は、スーパーコンピュータのそれと大差ないものであった。バッチ処理でこそないものの、シングル・ユーザでの利用を想定したものが多く、マルチ・ユーザであっても空間分割による利用が主である。多くがバックエンドとしてのシステムであり、クラスタ自体とは別にフロントエンドとなるワークステーション (WS) が必要となる場合が多い。つまりクラスタは、物理的形態こそ分散システムではあるが、それ以外の性質はまだ集中システムのそれに近いものであると言える。

また、分散コンピューティング分野からのアプローチにおいては、主にソフトウェアオーバヘッドに起因する通信コストの問題によりネットワークのバンド幅を有効利用できず、ネットワークで結合された計算機資源の有効利用を実現するまでには至っていない。また従来の分散システムの流れを汲むため、プロセスレベルのマルチ・ユーザ環境はある程度実現されているものの、システム全体にわたるスケジューリングは不十分である。

我々の研究室では、ネットワーク分散型並列処理システム「Computer Colony」の実現を目指している。Computer Colony では既存の分散システムに対しハードウェア、ソフトウェア両面から新規開発を行なうことにより、従来問題であった通信コストとマルチ・ユーザ環境への対応不足の問題を解決する。また、LAN 程度の分散環境においてネットワークに接続された計算機資源を、並列アプリケーションの実行によって有効利用することにより、高い性能とマルチ・ユーザにとって快適な環境を提供する。具体的には、分散環境において比較的細粒度の並列処理を可能にする高速、低遅延かつ信頼性のある通信を行なう通信機構を設計し、その機構を実現するネットワーク・インタフェースを新たに製作する。また、システムを利用する全てのユーザに対してシステム内の資源の公平な分配と位置透過性を実現する分散オペレーティングシステムを新たに開発する。

本稿ではこのうち通信コストの問題を解決し、分散環境において比較的細粒度の並列処理を可能にする通信機構について述べる。本通信機構では、分散共有メモリベースの通信を行ない、リモートメモリのキャッシングによる通信の最適化, ユーザレベル通信, 軽量通信プロトコルによる信頼性保証, デッドロック対策を行なうことにより、従来の分散システムでは不可能であった比較的細粒度の並列処理を実現する。また、通信機構を実装したプロトタイプ・ハードウェアとネットワーク・インタフェースの構成法を示し、予備的評価によってその有効性を示す。

本稿の構成は以下の通りである。まず、2において我々の研究室で開発中のネットワーク分散型並列処理システム「Computer Colony」を示し、ネットワーク分散型並列処理システムの定義と、実現のための要件について述べる。次に3でネットワーク分散型並列処理システム実現のために必要な通信機構について、その目標と解決すべき問題点、種々の通信機構を示し、問題解決のための本通信機構のアプローチについて述べる。4では、3で述べたアプローチに基づいて設計を行なった各機構について説明する。5では、上記の通信機構を実装した Computer Colony のプロトタイプ・ハードウェアとネットワーク・インタフェースを示し、予備的評価によってその有効性を示す。最後6においてまとめを行なう。

2 Computer Colony

本章では、ネットワーク分散型並列処理システム Computer Colony について、その目標、システムモデルと実現のために必要な技術を示し、分散システムにおける並列処理とはどのようなものであるか明確にする。

2.1 Computer Colony の目標

我々の研究室では、ネットワーク分散型並列処理システム「Computer Colony」の開発を行なっている。Colony —— 群体とは、同種の生物個体が多数集まって共通の体を組織し、相互に連絡して生活するものをいう。すなわち Computer Colony とは、「複数の独立した Computer からなり、全体として 1 つの Computer のように動作するシステム」を意味する。

Computer Colony は、現在の分散システムを拡張したシステムである。その目標の一つは、ネットワークによって分散した計算資源を有機的に結合し、現在の分散システムでは実現できない高性能を実現することである。また Computer Colony では、高性能を実現しつつ、従来の分散システムのような利用形態を実現し、快適性を維持することも目指す。これらの目標は、分散システム

において以下の 5S を満たすことにより実現される。

1. **S**calability : マルチプロセッサシステムとして理論上同等性能を持つ MPP に比肩する並列処理性能を持つ。また計算資源の追加によってリニアな性能向上が得られる。
2. **S**eamless Distribution : システムを利用する複数のユーザは手元の端末で Colony 内の全ての計算資源を、それらの位置を意識することなく利用できる。
3. **S**elf-Reconfiguration : システム構成の変更は Hot Plug で行なえ、追加削除された資源はシステムによって自動的に分配される。
4. **S**elf-Tuning : システム内の各アプリケーションのデータや計算の位置はシステムによって自動分配される。
5. Fair-**S**hare : システムを利用している各ユーザに対し、システムによる資源の fair な分配を行なう。

2.2 実現のための技術

Computer Colony の目指すネットワーク分散型並列処理システム実現のためには、従来の分散システムに対してハードウェア, オペレーティング・システム (OS) , プログラミング・システム等のいくつかの分野で新たな技術が必要になる。

ハードウェア : ネットワーク分散型並列処理システムでは、MPP に比肩する並列処理性能を実現する。現状の分散システムにおいても、利用するプロセッサの性能は MPP と変わらない。決定的に異なるのはシステムを構成するノード間の通信性能である。ネットワーク分散型並列処理システムを実現するには、ネットワークで接続されたノード間での効果的な並列処理が行なえるような通信機構を実現するハードウェアが必要である。

OS : 先の 5S のうち、Scalability 以外の目標は主に OS の役割である。これらは主にシステムを利用する全てのユーザが快適, 公平にシステムを利用できるように資源管理を行なうことにより実現される。ユーザに対してネットワークで接続されたノード全体を一つの Computer として提供するネットワーク分散型並列処理システムでは、従来の分散システムより一層のシステムワイドな、またマルチユーザ向けのスケジューリングを行なう OS が必要である。

プログラミング・システム : 並列処理環境を利用する並列アプリケーションが必要である。そのためには、並列化コンパイラ, 並列プログラム技術または言語等のプログラミング・システムの進歩が必要であろう。ただしネットワーク分散型並列処理システムにおいて特に必要とされる技術は、現在研究されている並列計算機向けの技術とは別に、プロセッサ間の距離が物理的に遠い分散環境向けの技術であろう。

Computer Colony ではこのうちハードウェアおよび OS の新規開発というアプローチをとる。ハードウェア技術としては、従来よりも高速, 低遅延な通信機構とそれを実現する通信ハードウェアの開発、OS としては Computer Colony 専用の分散 OS Colonia の開発を行なっている。

2.3 システムモデル

先の目標を実現するシステム構成として、i-Cartridge と Rack による Computer Colony のモデルを図1に示す。

2.3.1 構成

Computer Colony の最も重要な構成要素は、「**i-Cartridge**」(individual :個体) と呼ぶカートリッジである。 i-Cartridge は、プロセッサとキャッシュ(それぞれ複数あってもよい), それに主記憶, ネットワーク・インターフェイスを内蔵し、1 つの Computer として動作可能なものである。

i-Cartridge を収める **Rack** は、i-Cartridge を収める Bay, 収納する i-Cartridge 間を結ぶ **Intra-Rack Network**, システム内の Rack 間を結ぶ **Inter-Rack Network** へのインターフェイス, その他の I/O スロット, 電源装置を内蔵する。 デスクトップ・モデルでは 1 ~ 4 個程度、サーバ・モデルでは数十 ~ 数千程度の i-Cartridge Bay を持つ。Inter-Rack Network で接続されたシステム全体を **Colony** と呼ぶ。

Colony は非均質であってよい。 Colony 内にはいろいろな規模や構成の Rack を混在させて構わない。 また Rack 内に異なる構成の i-Cartridge を混在させてもよい。 ただし、大きな Rack を用い、Rack 内には同じ構成の i-Cartridge を揃えた方が並列処理性能が向上するであろう。 しかしもちろん、そうでなければ並列処理ができないということはない。

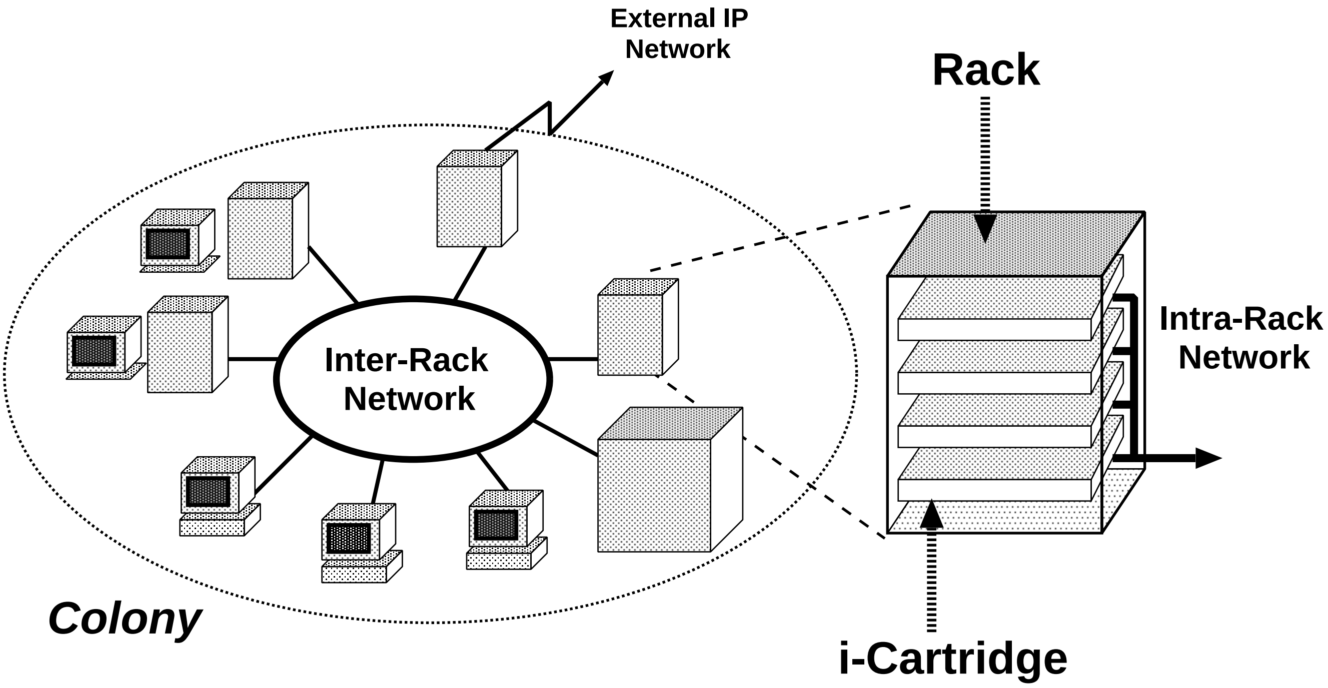

図 1: Computer Colonyのモデル

2.3.2 動作モデル

Computer Colonyでは、Colony全体にわたって単一のオペレーティング・システム(OS)が動作し、Colonyが1つのComputerとして動作する。その動作モデルは以下のようなものである。

- ユーザは、手元の端末がその資源を内蔵しているかのようにColony内の全ての計算資源を利用することができる。
  - ジョブはColony内部の適当な場所で実行される。インタラクティブなジョブはデスクトップのRackの中で実行されるだろう。巨大な並列ジョブはColony内の適当なRack、場合によってはColony内の全てのRackにまたがって実行される。ちょうど、SMPでどのプロセッサが働いているか分からないように、ユーザはジョブがどのRackのどのi-Cartridgeで実行されているか知らなくてよい。速いi-Cartridgeや空いているRackを探してリモートログインしたり、そのマシン名を打ち込んだりする必要はない。そもそも管理者以外のユーザは、個々のRackやi-Cartridgeの名前を知る必要はない。
  - OSを格納した不揮発性記憶装置ーーーハードディスク・ドライブは、全体で1つあればよい。個々のRackはディスクレスであってよい。
  - ファイルシステムも全体で1つである。アプリケーションはColony全体に対して1度インストールすればよい。データはシステムによって個々のi-Cartridgeに適当にキャッシュされる。
  - 外部のネットワークーーーIPネットワークへは、RackのI/Oスロットを通して接続する。Rackの1つ1つを外部へ接続する必要はない。普通、Colony全体で1つのポートと、1つのIPアドレスがあれば十分であろう。
- 資源はシステムによって自動的に、各ユーザにfairに分配される。他人のCPUバウンド・ジョブによって自分のインタラクティブ・ジョブの実行が阻害されることはなく、ほとんどのユーザはバックグラウンドでジョブが走っていることに気が付きさえしない。
- 新たに購入したi-Cartridgeは、Rackの前面にあるBayにHot Plugすれば自動的にColony内に組み込まれ、その分だけColonyの性能が向上する。Bay

が足りなくなれば、Rack を追加、またはアップグレードする。その場合にも全体の電源を落としたりする必要はない。

3 ネットワーク分散型並列処理システムを実現する通信機構

本章では、ネットワーク分散型並列処理システム実現のために必要な通信機構の位置づけを明らかにするため、分散環境における通信の問題について述べた後、関連する種々の通信機構と本通信機構の問題解決のためのアプローチを示す。

3.1 分散環境における通信の問題

本通信機構の目標は、従来の分散システムで問題であった通信コストを削減し、LAN 規模の分散システムにおいて比較的細粒度の通信を実用可能なレベルで実現することである。このことにより並列アプリケーションを効果的に動作させることを可能にし、分散している計算資源の有効利用と MPP に比肩する性能を実現する。

そのためには、分散環境において通信を行なう際に発生する以下のような問題を解決する必要がある。

レイテンシ, スループット不足 : 従来の分散システムでは通信時のレイテンシが非常に大きく、細粒度並列処理を行なうことは実用的ではなかった。その内容は、主に通信資源保護のためのシステムコールや、後述の信頼性保証のためのプロトコル処理などのソフトウェアオーバヘッドである。細粒度並列処理を実用レベルで実現するためにはこれらのオーバヘッドのほとんどを削減する必要がある。しかしこれらのレイテンシを削減しても、物理的にプロセッサ間の距離が遠い分散システムでは不可避なレイテンシが発生する。また、スループットも専用ネットワークを用いる MPP にはかなわないため、レイテンシの削減の他に通信の絶対量を減らす最適化が必要になる。この最適化は主にソフトウェアによってなされるもので、通信機構のハードウェアとしてはこの最適化をサポートするような機構が必要になる。

通信の信頼性の問題 : 汎用ネットワークではパケットの欠落が発生する可能性が比較的高い。汎用ネットワークのビットエラー率はおよそ $10^{-12} \sim 10^{-15}$ [1, 2] である。MPP は専用のネットワークを持ち、ノード間距離も短い。そのためエラー率は低く、パケットの欠落は uncorrectable-error として扱ってよかったが、分散システムにおいては適切な処理を行なう機構が必要になる。また汎用ネットワークにおいてはパケットの欠落のほか、ネットワークトポロジによってもパケットの重複, 追い越しが発生する可能性があり、パケットの point-to-point FIFOness が保証されない。

優先順位の不足 : 汎用ネットワークでは MPP の専用ネットワークのように通信路やバッファの多重化が通常なされない。そのため、デッドロックの発生についても考慮する必要がある。

3.2 種々の通信機構

分散環境において発生する上記の問題を解決すべく、様々な通信機構の研究が行なわれている。これらの通信機構は、メッセージ通信によるものと共有メモリによるものに大分することができる。以下では本通信機構のアプローチを明確にするため、メッセージ通信によるものと共有メモリによるものそれぞれについて提案されている数種の手法の得失をまとめる。

3.2.1 メッセージベースの通信機構

メッセージベースの通信機構は、分散システムの構成に自然な通信機構である。並列処理を目指すメッセージベースの通信機構では通信のオーバヘッドを減少させる通信ライブラリ、ファームウェアの開発が主に行なわれている。そのような通信ライブラリの例としては、AM [3], FM [4], VMMC-2 [5], PM [6] 等の myrinet [2] ベースのものが多い。また最近では GigaE PM [7] のように Gigabit-Ethernet ベースのものも研究されている。

図 2にメッセージベースの通信機構における処理の概略を示す。これらの通信機構では、通信はプログラム上で明示的に指定される。ホストプロセッサによって合成されたパケット識別子とデータは GigaE PM では軽微なシステムコールによって、また FM ではホストプロセッサによる Programmed I/O (PIO) によって、VMMC-2, PM では Direct Memory Access (DMA) によって通信ハードウェア上の送信バッファに転送され通信が行なわれる。AM はメッセージサイズが小さい場合には PIO、大きい場合には DMA を利用するというハイブリッドな方法をとり、レイテンシの削減を図っている。

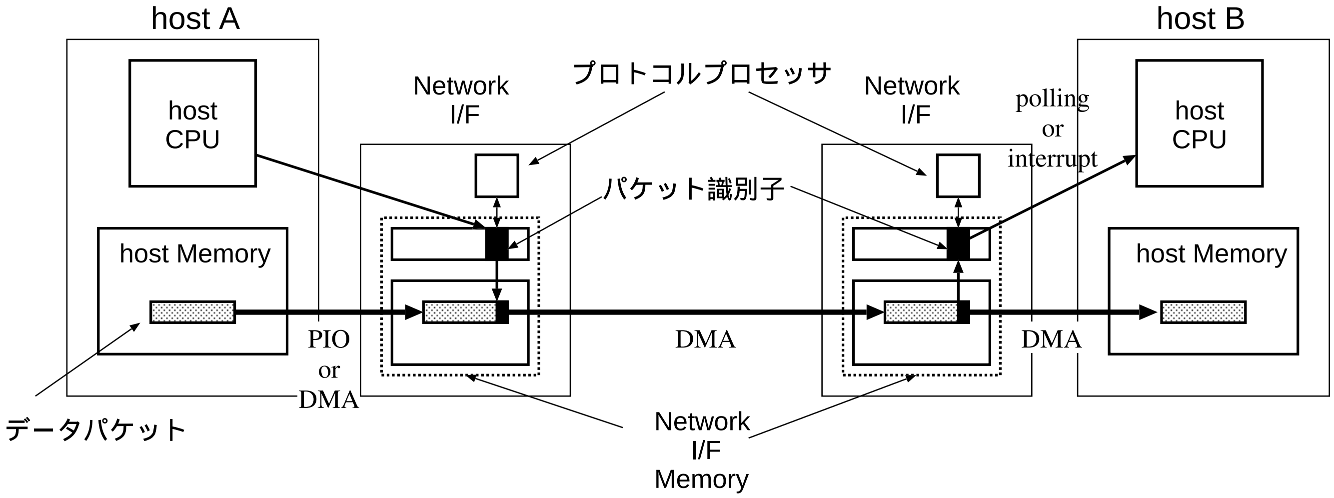

図 2: メッセージベースの通信機構の処理

これらの通信機構における通信の信頼性は、従来のTCP/IP のようにホストプロセッサの上位プロトコルで実現するものや、ネットワーク・インタフェースによって下位のデータリンクレベルで処理を行ない、レイテンシを削減を目指すものがある。ただし、このデータリンクレベルの信頼性保証処理はネットワーク・インタフェース上のプロトコルプロセッサにおいてソフトウェアによって行なわれる場合が多い。

従来のTCP/IP による通信のレイテンシが数百μsであったのに対し、これらの通信ライブラリではレイテンシは数十μsまで減少させることができる。

しかしメッセージベースの通信ではレイテンシの他に通信前のパケット識別子構成の処理やDMA の起動コストが大きい。さらに、ソフトウェアによる信頼性保証のためのプロトコル処理のオーバヘッドも依然存在する。また、アプリケーション製作者の立場から見れば、プログラム中で明示的に通信を指定する必要があるため、アプリケーションによっては並列化が困難な場合がある。

よってメッセージベースの通信機構では、比較的粒度の粗い並列処理は可能であるものの、細粒度の並列処理を行なうのはやはり現実的ではない。

3.2.2 分散共有メモリベースの通信機構

ハードウェア分散共有メモリベースの通信機構はMPP～SMP クラスタの形態をとるシステムに多い。そのような通信機構の例として、Typhoon [8], Synfinity [9], Memory Channel [10] を取り上げる。

Typhoon Wisconsin 大学 Madison 校のTempest/Typhoon は、共有メモリ、メッセージ通信、またはそれらを融合した並列アプリケーションを、ワークステーションクラスタ、メッセージ通信型並列計算機、共有メモリ型並列計算機などの様々なプラットホーム上で効果的に実行するためのインタフェースである。

Tempest では、共有メモリ管理のため、Stache と呼ばれるポリシーを利用する。Stache ではノードのローカルメモリの一部にリモートデータのキャッシングを行ない、ネットワークのトラフィックを減少させる。また共有のためのマッピングはページ単位で行なうが、コヒーレンス制御はキャッシュブロック単位で行なう。そのため、ローカルメモリのブロックごとのタグを利用している。

Typhoon はこれらの機構をハードウェアサポートによって実現する実装である。Typhoon では、既存のワークステーションのシステムバスにNP(network interface processor) というネットワーク・インタフェースを接続する構成をとる。NP にはプロトコルプロセッサが装備され、通信に関わる処理はこのプロセッサが行なう。

Synfinity HAL Computer Systems, Inc. のHAL S1 System は、富士通で開発されているSynfinity interconnect を用いた共有メモリ、メッセージパッシングマシンである。主にintel 製プロセッサを4 つ搭

載したサーバ同士を接続する。ネットワークは専用のSynfinity interconnectである。ネットワーク・インタフェースにタグ用のSRAMとMCU(Mesh Coherent Unit)と呼ばれるをコヒーレンス制御機構を備え、最大4ノードのクラスタ内でハードウェア共有メモリを実現する。

MCUの主な機能は、グローバルでキャッシュコヒーレントなアドレス空間を提供することである。この機能は、バスベースのスヌープ方式で行なわれ、そのためのディレクトリとして各ノードのSRAMを利用する。sparseディレクトリと呼ばれるこのディレクトリは、自ノードの主記憶内のブロックの内、他ノードのプロセッサにキャッシュされているブロックの情報のみを持つ。そのため、主記憶内の全てのキャッシュブロックのタグを持つディレクトリに比べて、容量がかなり小さくて良い。

MCUでは、低レイテンシな24Byte程度の小さなメッセージ通信や、DMAを利用し高バンド幅を実現するサイズの大きいメッセージ通信もサポートする。

Memory Channel Memory Channel Networkでは、クラスタ間の通信レイテンシを5μs以下、オーバヘッドを0.5μs以下にするという目標のため、リフレクティブメモリのコンセプトをクラスタ間通信用に拡張したアーキテクチャを採用している。リフレクティブメモリとは、システム内のあるメモリ上にライトが発生した場合、そのメモリと関連づけられた全てのメモリに対して瞬時にライトが反映される、というものである。

Memory Channel Network for PCIでは、この関連づけをページ単位で行ない、送信側のI/O空間と受信側の物理メモリの間に単一のネットワークアドレスを用いてコネクションを確立する。Memory Channel for PCIでは、新たに通信を開始する際に、あらかじめ通信のためのコネクションを確立しなければならない。

コネクションの確立は、アプリケーションがクラスタサービスというソフトウェアを利用して、PCIバスに装着されるネットワーク・インタフェース上のpage control table(PCT)の設定をすることにより行なわれる。

PCTでは、ネットワーク・インタフェースのI/O空間とMemory Channelネットワークの空間であるネットワークアドレス空間をダイレクトマッピングする。さらにPCTでは、ネットワークアドレス空間内の全てのページに対して、自ノードではどのように振る舞うかを規定する情報を保持する。

一旦コネクションが確立されれば、送信側ではI/O空間に対するライト、受信側では物理メモリに対するDMAのみで通信が行なわれる。

これらの共有メモリベースの通信機構では、プログラム中に明示的に通信を指定する必要はない。また、通信はメモリアクセスによって起動され、ネットワーク・インタフェースでメモリアクセストランザクションを元にパケットが構成される。これらの通信機構のほとんどは専用のネットワークを持ち、デッドロック対策もなされている。これらの通信機構ではレイテンシが数μsと小さく、細粒度の並列処理が可能である。

しかし、専用ネットワークを用いるこれらの通信機構ではネットワークの信頼性は十分高いと仮定し、特に信頼性保証処理は行なわれないものがある。また、ノード数に関して制限を課しているものもある。さらに、専用ネットワークの開発、導入コストも大きなものとなる。

3.3 本通信機構のアプローチ

本通信機構は、上記のメッセージベース、ハードウェア共有メモリベースの両者を融合した通信機構である。すなわち、「信頼性保証処理を行なう汎用ネットワーク」を用いて「ハードウェアによるキャッシュコヒーレンス制御を行なう分散共有メモリベースの通信」を実現する。

そのため、汎用ネットワークを利用することにより発生する信頼性やデッドロックの問題を解決しつつ通信処理のレイテンシを削減し、汎用ネットワークを用いた分散システムにおいて細粒度の並列処理を実用レベルまで高めることを目指す。

本通信機構では以下の4つのアプローチによってこれを実現する。

1. キャッシングによる通信最適化のサポート

   汎用ネットワークを利用することに起因するレイテンシ, スループットはMPPの専用ネットワークにはかなわない。このため、通信の絶対量を減らす最適化が必要になる。そこで、メインメモリをリモートメモリの大容量キャッシュに利用できる機構を提供する。この機構を実現するために、ネットワーク・インタフェースをシステムバスに接続し、さらにネッ

トワーク・インタフェース上にメインメモリのキャッシュタグを備え、コヒーレンス制御を行なう。

2. ユーザレベル通信

   従来の分散システムで必要だった通信の際のシステムコールをなくすユーザレベル通信を実現する。そのためシステムバスに接続したネットワーク・インタフェースを利用し、従来システムコールで行なわれていた、通信開始処理と通信資源のプロテクションを実現する。

3. 軽量通信プロトコル

   通信プロトコルを簡略化し、ネットワーク・インタフェース上で信頼性保証処理を行なう。さらに、従来プロトコルプロセッサで行なわれていた逐次的なソフトウェア処理に対し、ハードウェアサポートを行なう。このことによりプロトコル処理全体に関わるオーバヘッドを削減する。

4. デッドロック対策

   ネットワーク・インタフェース上の送受信バッファの多重化を行なうことに加え、また信頼性保証処理に利用する再送バッファを流用することによって、デッドロックを防止する。

4 通信機構の設計

本章では、汎用ネットワークを使用して構成した分散システムにおいて、細粒度並列処理を行なう際に発生する問題を解決するための、本通信機構による 4 点のアプローチについて説明する。この 4 点のアプローチとは、通信最適化のサポート, ユーザレベル通信, 軽量通信プロトコルによる信頼性の保証, デッドロック対策である。

4.1 キャッシングによる通信最適化のサポート

本通信機構では細粒度の並列処理を可能にするため、ハードウェアによるコヒーレンス制御を行なう分散共有メモリベースの通信を実現する。

共有メモリを利用した通信においては、リモートメモリのキャッシングを行なうことにより、無駄な通信回数を減らす通信の最適化が可能になる。この方法はノード間の通信コストがノード内の通信に比べて非常に高くなる分散システムベースの構成では大変有効である。

本通信機構では、リモートメモリのキャッシングによる通信の最適化をより効果的に行なうため、メインメモリの一部をリモートメモリのキャッシュとして利用する。また、ネットワーク・インタフェースをシステムバスに接続することによってハードウェアによるコヒーレンス制御を可能にし、コヒーレンス制御に関わるソフトウェア処理のオーバヘッドをなくす。

以下、そのための分散共有メモリの構成と、ネットワーク・インタフェースによるリモートメモリキャッシュのコヒーレンス制御について述べる。

4.1.1 分散共有メモリの構成

物理的に共有メモリを持たない分散システムにおいて共有メモリを実現する手法として、仮想共有メモリ (Shared Virtual Memory : SVM) [11] がある。この手法では、ネットワーク上に一つの仮想空間を構成し、物理的に分散したメインメモリをその仮想空間にマッピングすることによって共有メモリを実現するものである。

この時、ノード間でのデータの共有単位としては、ページ単位とキャッシュブロック単位が考えられる。ページ単位の場合、データ転送、コヒーレンス制御を行なうと、false-sharing が発生してしまい、通信が頻発する。一方、共有の単位をキャッシュブロックの単位とすると、データの存在するノード、アドレスを管理するディレクトリの容量が爆発的に大きくなってしまう。

本通信機構では超並列マシン JUMP-1 で採用されている方式 [12] を採用する。リモートメモリアクセスが発生した場合、アクセスした側のメモリ領域の確保はページ単位で行なうが、実際のデータ転送はブロック単位で行なう。

この方式により、ディレクトリの容量を抑えつつ、false-sharing による無駄な通信の発生をなくすことができる。

本通信機構では、OS によって管理された以下の物理ページを利用してメインメモリの一部にリモートメモリのキャッシングを行なう。

- ホームページ

  物理メモリが実装されている物理アドレスに位置し、メモリ上にデータの実体が存在する。ホームページが存在するノードを、その仮想ページのホームノードと呼ぶ。

- ダミーページ

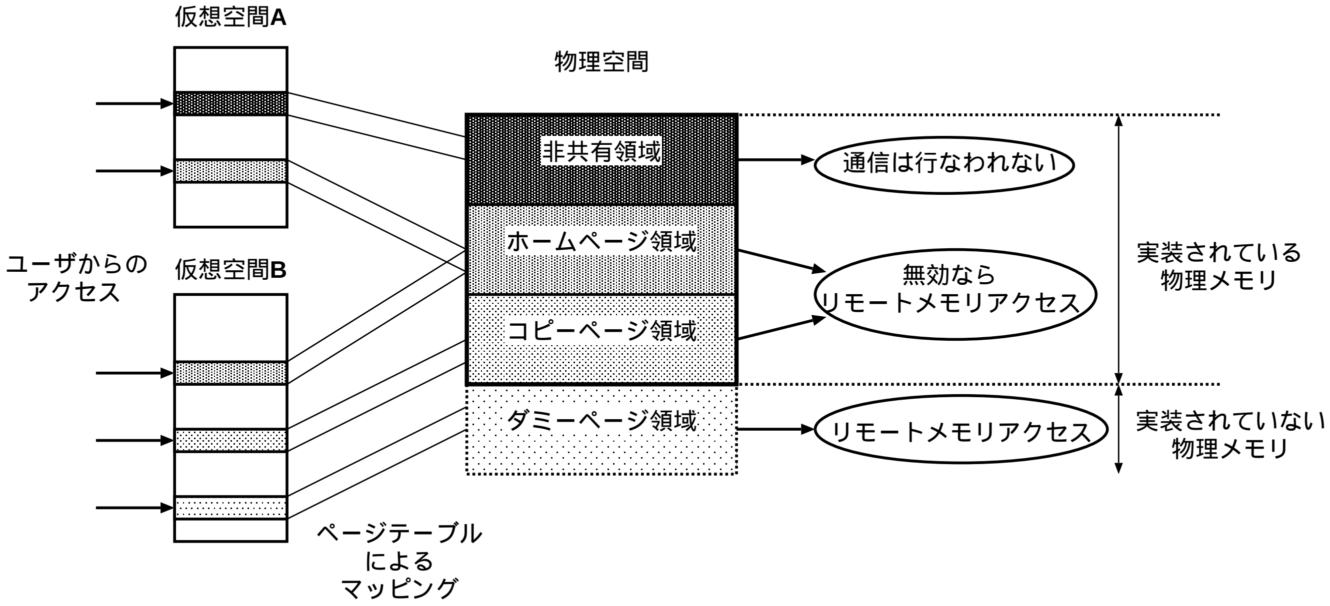

図 3: 物理ページの分類

物理メモリが実装されていない物理アドレスに位置し、直接のアクセスはできない。ダミーページに対するアクセスがあると、対応するホームページへリモートメモリアクセスを行なう。得られたデータは当該ノードのメモリにはキャッシングしない。

- コピーページ

  物理メモリが実装されている物理アドレスに位置し、ホームページの内容をキャッシングする。したがって、ホームページとの間で適切な一貫性制御を行なう必要がある。内容が無効であるときにアクセスすると、ダミーページにアクセスした場合と同様、リモートメモリアクセスを行なう。

共有領域の確保は、基本的にはページテーブルにおいて、共有する仮想ページをダミーページにマッピングすることによって行なう。またページ全体に対して頻繁にアクセスされるような場合はコピーページにマッピングし、ノード内のメモリにキャッシングする。ただし、キャッシングするデータの転送はブロック単位で行なう。

4.1.2 システムバスを利用した通信

本通信機構では図 4のようにシステムバスにネットワーク・インタフェースを接続し、ネットワーク・インタフェースがキャッシュコヒーレンス制御を行なえるような構成を採用する。

従来の分散システムでは、ネットワーク・インタフェースはI/O バスに接続されるものであった。しかし従来通りI/O バスにネットワーク・インタフェースを接続した場合、リモートメモリのキャッシングを高速に行なうためのハードウェアによるコヒーレンス制御が行なえない。なぜなら、I/O バスに接続されているネットワーク・インタフェースからは、バス・ブリッジを経由するため、システムバスに接続されているプロセッサのキャッシュを直接操作できないためである (図 4(a))。そのためI/O バスにネットワーク・インタフェースを接続すると、共有メモリによる通信の際ソフトウェアによるコヒーレンス処理が発生してしまい、効率の良いリモートメモリのキャッシングが行なえない。

対称型マルチプロセッサ (Symmetric Multi Processors : SMP) をサポートしている WS/PC では、一つのノード内に複数のプロセッサが存在し、一般にノード内のプロセッサ間ではハードウェアによるキャッシュコヒーレンス制御が行なわれる (図 4(b))。

そこで、コヒーレンス制御を行なうハードウェアとしてネットワーク・インタフェースをシステムバスに接続すれば、ノード内のプロセッサのキャッシュに対する操作が可能になる (図 4(c))。このため、ネットワーク・インタ

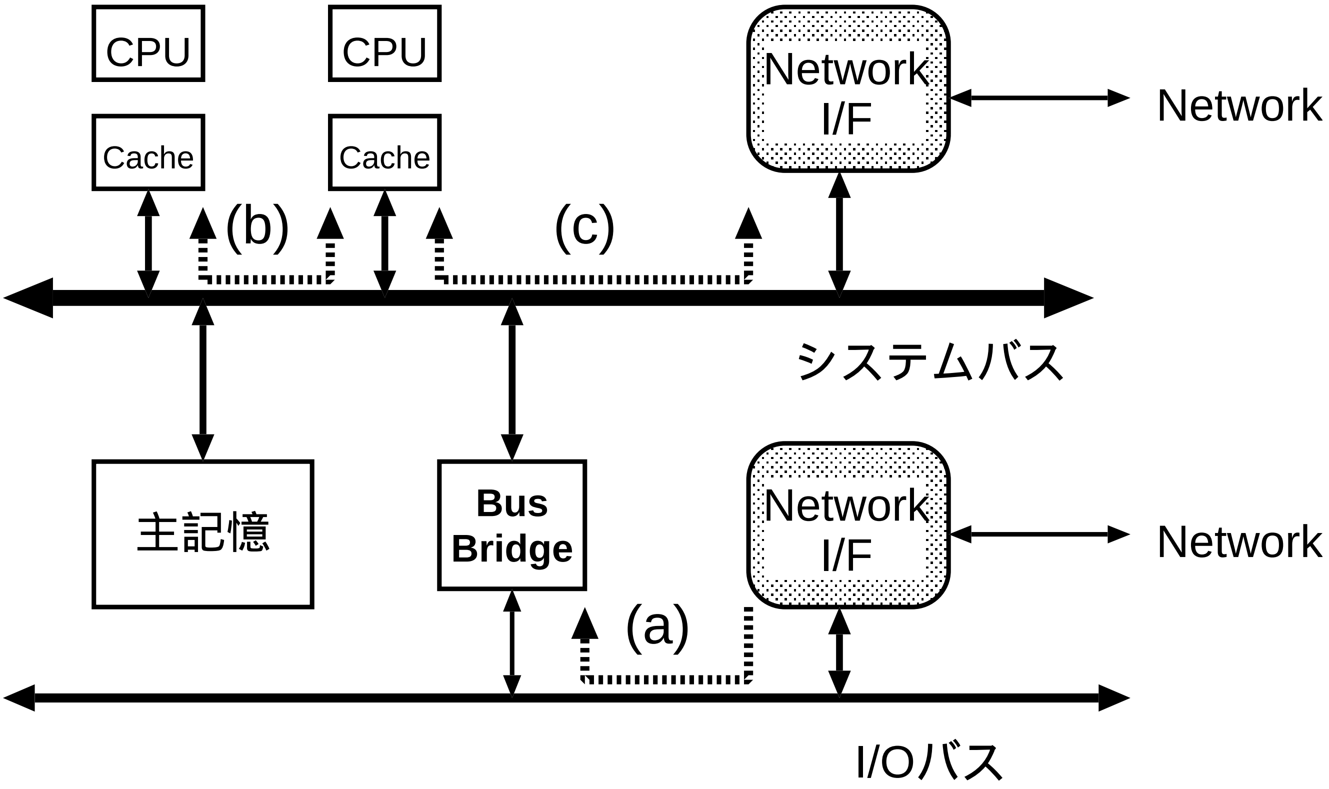

図 4: ネットワーク・インタフェースによるコヒーレンス制御

フェースを通して他ノードからのプロセッサのキャッシュに対する操作が行なえるようになり、ノード間のキャッシュコヒーレンス制御が可能になる。

4.1.3 主記憶タグによるコヒーレンス制御

本通信機構では、システムバスに接続したネットワーク・インタフェースによってブロック単位のキャッシュコヒーレンス制御を行なう。そのため、ネットワーク・インタフェース上にメインメモリのブロック単位のキャッシュディレクトリを持つ。本通信機構では、このディレクトリを主記憶タグと呼ぶ。コヒーレンスプロトコルが一般的な write-invalidate 型の場合には、主記憶タグの保持する情報は以下のようなものである。

共有状態のタグ 当該ブロックが共有されているかどうかを示す。共有されているブロックに対するアクセスがあった場合には、アクセスを行なったプロセッサに対してその旨通知する。

有効性のタグ 当該ブロックが有効かどうかを示す。無効なブロックに対するアクセスを関知すると、そのアクセスを中止させる。そのブロックの有効なデータをネットワーク・インタフェースは責任を持って提供する。

ネットワーク・インタフェースはシステムバス上の全てのトランザクションをスヌープし、トランザクションのアドレスとそのアドレスに対応する主記憶タグの内容によって適切な処理を行なう。

4.2 ユーザレベル通信の実現

ユーザレベル通信とは、通信時に OS が関与せずに行なえる通信である。従来の分散システムでは、通信の度にシステムコールが必要であった。このシステムコールの役割は、主に通信資源の保護や通信イベントの通知であり、そのための処理やコンテクスト・スイッチが通信時のオーバヘッドとなっていた。

本通信機構ではこのオーバヘッドをなくすため、システムバスに接続されたネットワーク・インタフェースを利用する。

ネットワーク・インタフェースによってユーザレベル

通信を行なうためには、システムコールで実現されていた通信資源の保護を行なう必要がある。また、ネットワーク・インタフェースが自律的に通信の必要性を関知し、必要な処理を行なわなければならない。

本通信機構では、以下のようにしてこれらの要件を実現する。

4.2.1 通信資源のプロテクション

本通信機構ではシステムバスに接続したネットワーク・インタフェースによって通信を行なう。システムバスを利用した通信では、仮想記憶管理のためのMMU(Memory Management Unit, メモリ管理ユニット) によるメモリ保護機構を利用することで、通信機構をユーザに対して安全に解放することができる。すなわち、通信時にユーザが行う仮想アドレスでのメモリアクセスを、MMU が物理アドレスでのメモリアクセスに変換する。この物理アドレスへの変換により、通信資源が通常のメモリアクセスと同様に保護される。

4.2.2 自律的な通信の起動

本通信機構ではネットワーク・インタフェースが自律的にリモートメモリアクセスを関知して通信起動の処理を行なう。

システムバスにネットワーク・インタフェースを接続する本通信機構では、リモートメモリアクセスの必要なブロックを示す物理アドレスを検出することにより通信処理を開始する。

MMU によって変換されたシステムバス上の物理アドレスは、ホームページ, コピーページ, ダミーページのどれかに属する。ネットワーク・インタフェースは、その物理アドレスの属するページの種類と、その物理アドレスの示すブロックの状態によって通信の必要性の判断を行なう。

- ホームページまたはコピーページに属する物理アドレスの場合 :主記憶タグを調べ、無効なブロックに対するアクセスであった場合のみリモートメモリアクセスを行なう。
- ダミーページに属する物理アドレスの場合 :当該ブロックは物理メモリ上にないため、必ずリモートメモリアクセスを行なう。

4.2.3 ネットワークアドレス変換

本通信機構では、先に述べたようにメモリアクセス時にシステムバス上を流れる物理アドレスによって通信を起動する。

しかし、リモートメモリアクセスを行なう際には、この物理アドレスをそのまま利用することはできない。なぜならこの物理アドレスによって指定される物理メモリはローカルノード内で閉じたものであり、リモートノードにおける物理メモリとは関連のないものだからである。

よってネットワーク上では仮想的なネットワークアドレスが必要となる。この場合、ネットワーク・インタフェースで物理アドレスとネットワークアドレスを変換する機構が必要になる。ネットワークアドレスとしてどのようなものを利用するかは任意性があるが、本通信機構では仮想空間ID + 仮想アドレスを用いることとする。

ネットワークアドレスとローカルな物理アドレスの変換を行なうため、ネットワーク・インタフェースが仮想アドレス→物理アドレス変換のためのページテーブルや、物理アドレス→仮想アドレス変換のための逆引きページテーブルを参照することが必要になる。

しかしこれらのテーブルは通常メインメモリに存在するため、ネットワーク・インタフェースが通信の度にメインメモリ上のテーブルにアクセスを行なうとネットワーク・インタフェースによるシステムバスの占有時間が増加し、プロセッサからのメインメモリアクセスを阻害してしまう。また、ネットワーク・インタフェース上に全てのテーブルを保持する大容量のメモリを実装するのは現実的でない。

そのため本通信機構ではメインメモリの一部をネットワーク・インタフェース上に必要に応じてキャッシングする機構を提供する。ページテーブル, 逆引きページテーブルの一部をキャッシングし、仮想アドレス→物理アドレス変換のためのTLB(Translation Lookaside Buffer), 物理アドレス→仮想アドレス変換のためのRTLB(Reverse TLB) として利用する。

ネットワークアドレスの変換は送信側、受信側で以下のように行なわれる。

- 送信側 :物理アドレスをキーとしてRTLB を引き、仮想空間ID + 仮想アドレスのネットワークアドレスを得る
- 受信側 :ネットワークアドレス内の仮想アドレス, 仮想空間ID をキーとしてTLB を引き、物理アドレスを得る

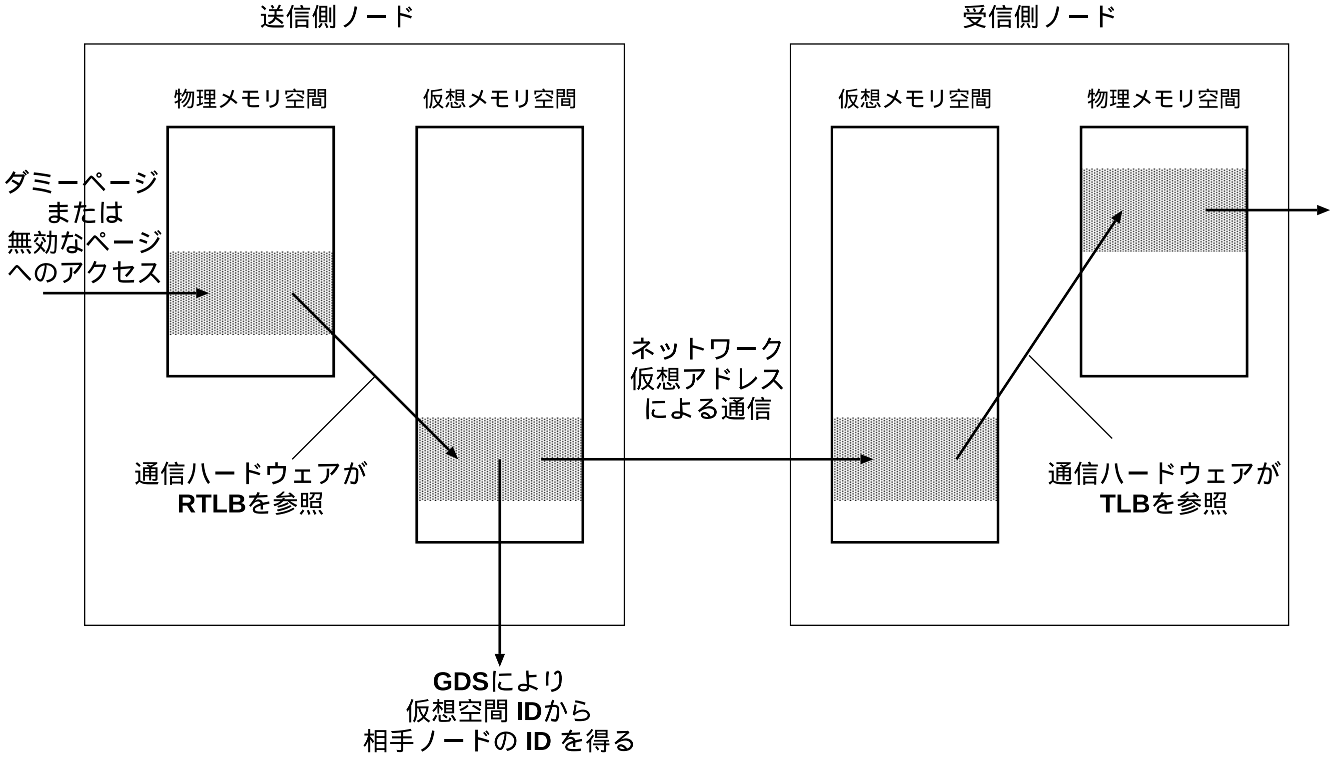

図 5: アドレス変換

4.2.4 通信先ノードの特定

本通信機構では、上記のネットワークアドレス変換機構の他、通信先のノードを特定するための機構が必要である。すなわちリモートメモリアクセスのアクセス先であるホームノードや、自ノードのメモリをキャッシングしているノードの位置を特定する必要がある。

このノード情報の管理はOSの役割である。Computer Colonyでは、専用OS Colonia上で提供されるサービスの1つであるGDS(Global Directory Service) [13] によってノード情報のディレクトリが用意される。

通信時にネットワーク・インタフェース上では、送信側においてネットワークアドレス変換で得られた仮想空間IDでGDSの用意したノード情報の入ったディレクトリを引き、通信先のノードIDを得る処理が行なわれる。

以上のアドレス変換に関わる動作をまとめると、図5のようになる。

4.3 軽量通信プロトコルによる信頼性保証

本通信機構では、通信の際のソフトウェア・オーバヘッドの一部であるプロトコル処理のオーバヘッドを削減するため、従来上位プロトコルで処理されていた信頼性保証のための再転送つき確認応答処理をネットワーク・インタフェース上で行なう。

また従来のプロトコル処理に対してデータリンクレベルでの実現のために処理を一部変更し、さらにそれらの処理に対してハードウェアサポートを行なう。このことにより、通信の信頼性を保証しつつ、処理の高速化と処理のオーバラップによってさらなる通信時のオーバヘッド削減を実現する。またハードウェアサポートにおいては、特に正常通信時のオーバヘッド削減を目指す。

ネットワーク・インタフェース上で行なう処理は以下のようなものである。

1. シーケンス番号によるパケットの順序管理
2. 再送バッファによる送信済パケットの管理
3. NACKによる再送をメインとする再送処理
4. Creditによるフロー制御

以下それぞれについて説明を行なう。

4.3.1 パケットの順序管理

汎用ネットワークでは、ネットワークの信頼性が比較的低いため、パケットの欠落が発生する。このパケットの欠落は通常のパケットのみならず確認応答 (ACK) パケットにも発生し得るので、パケットの重複および追い越しについて注意した順序管理が必要である。

従来の手法では、上位プロトコル処理の段階で送信パケットに一意なシーケンス番号を割り当て、確認応答にそれを含めて返送する。コネクション型の通信の場合は、現在確立されているコネクション内に閉じたシーケンス番号を用いることができるため、その管理は単純である。

本通信機構では、この順序管理を下位のデータリンクレベルで行なう。しかし、データリンクレベルのコネクションレス型通信の場合には、従来のようにあるノードから送信される全てのパケットに対して送信先を区別せずに sequential な番号づけを行なった場合、受信側でのパケットの順序管理の判断が困難になる。例えば、あるノード A が sequential な番号づけを行なった 3 つのパケット pkt1,pkt2,pkt3 をそれぞれノード B, ノード C, ノード B に送信した場合、ノード B では pkt2 が欠落したのか、それとも他のノードへのパケットだったのかを判断することは困難である。

そのため本通信機構では、送信側では送信先ノードごとにシーケンス番号を管理する。また、受信側では送信元のノードごとに次に受信すべきパケットのシーケンス番号を管理する。

この管理のため、送信側には送信先ノード番号ごとのシーケンス番号を格納するメモリ (送信側シーケンスメモリ)、受信側には送信元ノード番号ごとのシーケンス番号を格納するメモリ (受信側シーケンスメモリ) を設ける。

これらを利用した以下のような処理により受信パケットの順序管理を行なう。

送信側でのシーケンス番号割り当て 送信側では送信パケットに対して送信先ノードごとのシーケンス番号が割り当てられる。この時の処理は以下のようになる。

1. 送信先ノードに対応する送信側シーケンス番号を読み出し
2. 出力を送信パケットヘッダのシーケンス番号にセット
3. 出力をインクリメントしシーケンス番号を書き戻し

受信側でのシーケンス番号認識 受信側では受信パケットの順序管理のため、受信側シーケンス番号を利用する。受信では全ての相手ノードの受信側シーケンス番号を別々に保持し、パケットの追い越し、重複を認識する。受信したパケットのシーケンス番号 (pkt_seq) とそのパケットの送信ノードに対応する次に受信すべきシーケンス番号 ($next\_seq$) を比較し、確認応答の種類を決定する。

- $pkt\_seq = next\_seq$:正常受信 → ACK
- $pkt\_seq < next\_seq$:重複の発生 → ACK
- $pkt\_seq > next\_seq$:追い越しの発生 → NACK_seq

受信側では同時にチェックサムも調べる。

- エラーなし → ACK
- エラー発生 → NACK_checksum (追い越しの発生の場合は NACK_seq)

この時の処理は以下のようになる。

1. 受信側シーケンス番号を読み出し
2. 出力を受信パケットヘッダのシーケンス番号と比較
   (a) 正常受信なら出力シーケンス番号をインクリメント
   (b) 重複, 追い越しなら出力シーケンス番号はそのままの値
3. 受信側シーケンス番号の書き戻し

確認応答の構成 確認応答パケットの種類が上記のように決定されると、確認応答パケットの構成を行なう。必要なのは以下の処理である。

- 受信したパケットヘッダと送信する確認応答パケットヘッダの S_ID,D_ID を入れ替える
- 確認応答パケットヘッダに確認応答の種類とそれによる確認応答のシーケンス番号 (ack_seq) をセットする
  - ACK の場合 :$ack\_seq \Leftarrow pkt\_seq$ とし、シーケンス番号 pkt_seqのパケットを正常受信したことを送信側に伝える

  - NACK_checksum の場合 :$ack\_seq \Leftarrow pkt\_seq$ とし、シーケンス番号 pkt_seqのパケットの受信に失敗したことを送信側に伝える
  - NACK_seq の場合 :$ack\_seq \Leftarrow next\_seq$ とし、本来到着すべきシーケンス番号 $next_seq$のパケットが到着しなかったことを送信側に伝える

以上のような処理により、データリンクレベルにおけるパケットの順序管理が可能になる。

4.3.2 送信済パケットの管理

送信パケットの管理とは、確認応答を待つ複数の送信済パケットを保持し、再送が必要な場合に適当なパケットを送出できるようにすることである。

再送に必要な送信済パケットは、確認応答がなされるまでメモリ上の再送用の領域に保持される。そのため従来の手法では、再送が必要かどうかに関わらず、通信のたびに再送用の領域に対するデータコピーが発生し、オーバヘッドになっていた。

そこで、本通信機構ではネットワーク・インタフェース上に大容量の再送パケット用バッファを用意する。このバッファを再送バッファと呼ぶことにする。

また、再送バッファのディレクトリを用意し、送信済パケットの管理のための処理をハードウェアによって行なうことを可能にする。

以下それぞれについて説明する。

再送バッファ ネットワーク・インタフェース上に再送バッファを用意しても、そのバッファへの登録処理を通信処理と別に行なった場合にはデータコピーのオーバヘッドは解消しない。そのため再送バッファへの登録は、パケットを送信バッファからネットワークに送信する際に同時に行なう。

登録の際のアドレスは、パケットを一意に認識できるものでなければならない。そのため、送信パケットの送信先ノード番号 (D_ID) + シーケンス番号 を適当にハッシングしたものを利用する。本通信機構では送信パケットの D_ID + シーケンス番号の下位ビットを利用する。

再送バッファのディレクトリ ディレクトリは再送バッファの有効性を示すものである。送信パケットが再送バッファに格納された時点で有効フラグをセットし、その送信パケットに対する確認応答を受信した時点で無効化する。ディレクトリのキーとしては、パケット送信時の登録にはパケットの D_ID とシーケンス番号の下位ビット、確認応答受信時の無効化にはパケットの S_ID とシーケンス番号の下位ビットを利用する。送信時の登録、確認応答受信時の無効化を同時に行なうことを可能にするため、デュアルポートメモリにより構成する。

- 登録
  1. パケット送信準備完了
  2. 送信パケットの D_ID + シーケンス番号の下位ビットをキーとしてディレクトリをリード
     (a) 0 (空き) → パケット送信
     (b) 1 (衝突) → 0 まで送信を待つ
  3. 1 をライトバック
- 無効化
  1. 確認応答パケット受信
  2. 確認応答パケットの種類を認識
     (a) ACK → 確認応答パケットの S_ID + シーケンス番号の下位ビットをキーとしてディレクトリに 0 をライト
     (b) NACK → 再送処理

この再送バッファとディレクトリによる管理によって、送信時の無駄なデータコピーは必要なく、また正常通信時にはディレクトリの無効化を行なうだけで済み、オーバヘッドを削減することが可能になる。

4.3.3 再送処理

従来の手法は、送信パケット毎にタイマを用意し、送信側ではパケットを送信するときタイマをスタートさせ、確認応答が到着する前にタイマが切れればパケットを再転送する、というものであった。しかし、システムを構成するノード数が多い場合には、全ての送信パケットに対してタイマを用意し、その管理を行なうには広大なメモリ領域が必要であり、ネットワーク・インタフェース上で実現するのは困難である。

そのため、本通信機構では全ての送信パケットに対してタイマを用意することはせず、再送は主に否定応答 (NACK) によって行なう。

NACK による再送は、あるパケットの欠落が発生した場合、次のパケットを受信することによってそれ以前のパケットの欠落を認識する、ということに基づいた処理である。

このNACK による再送処理のプロトコルはPM で行なわれているModified ACK/NACK 方式 [6] に若干の変更を加えたものである。主な変更点は以下のようなものである。

- データパケットとACK は一対一で対応する。Modified ACK/NACK 方式では1 つのACK によって複数のACK の処理を行なう場合があるが、処理の簡略化のためこれは行なわない。
- Modified ACK/NACK 方式ではフロー制御の機能も実現しているが、本通信機構では受信側の処理の単純化のため、再送プロトコルとは別にCredit によるフロー制御を行なう。
- Modified ACK/NACK 方式ではネットワークがreliable であるとしてパケットの欠落に対する処理は行なわないが、本通信機構では再送プロトコルレベルでパケットの欠落にも対処を行なう。

しかし、上記のNACK による再送だけでは完全な再送処理は行なえない。なぜなら、あるパケットが欠落した後にパケットが送られないような場合、すなわちあるノードに対する送信パケットが唯一であるか、あるノードに対する一連の通信の最後のパケットが欠落した場合には、次に到着するパケットによって欠落を認識できないため、再送の手段を失ってしまう。

そのため、そのようなパケットに関してはタイムアウトの手段を提供する必要がある。この機構を実現するため、全ての送信先ノードついて最後に送信したパケットのタイムスタンプを保持するメモリを用意する。従来のように全ての送信パケットに関してタイマを保持する必要がないので、必要なメモリ容量は全ての送信先ノードにつき1 エントリに抑えられる。

4.3.4 フロー制御

送信側が受信バッファの容量を越えてデータを送信した場合、通常そのデータは破棄されてしまうため、送信側ノードは受信側のバッファの状態を常に把握しておく必要がある。そのための機構としてはmyrinet 等で行なわれるSTOP and GO 方式 [2] や、TCP/IP で行なわれているsliding window 方式 [14]、Fibre Channel 等におけるCredit を利用した方式 [1] 等がある。

本通信機構で行なうコネクションレス型の通信の場合、ある受信ノードに対して同時に複数の送信ノードが存在するため、送信側ノードが受信側ノードのバッファの状態を逐一把握することは困難である。そのため、送信ノードと受信ノードの間に位置するネットワークスイッチのバッファと、送信バッファもしくは受信バッファ間でのバッファ間フロー制御が必要になる。

本通信機構で利用するネットワークスイッチは、全ての接続ノードに対して独立した送信バッファ、受信バッファを持つと仮定し、それらのバッファ間でフロー制御を行なう。そのための機構として本通信機構ではバッファ間Credit を利用した方式 [1] を採用する。利用するフロー制御用のパラメータは以下のものである。

- Buffer-to-Buffer Credit

  送信先バッファの受信可能なパケットのエントリ数
- Buffer-to-Buffer Credit_Count

  送信先バッファの未処理パケットに占有されているエントリ数
- Receiver_Ready プリミティブ

  バッファのあるエントリがパケット受信可能な状態になったことを示すプリミティブ

ネットワーク・インタフェースではこれらを利用する以下のハードウェアを用意する。

- Buffer-to-Buffer Credit 用レジスタ
- Buffer-to-Buffer Credit_Count 用カウンタ

これらのハードウェアにより以下のようにフロー制御を行なう。

1. パケット送信管理
   (a) Credit > Credit_Count → パケット送信可能
   (b) Credit = Credit_Count → パケット送信不可
2. Credit_Count 管理
   (a) パケット送信 → Credit_Count + 1
   (b) Receiver_Ready プリミティブ受信 → Credit_Count − 1
3. Receiver_Ready プリミティブ送信

- 受信バッファ内のあるパケットの処理が完了 → Receiver_Ready プリミティブ送信

このバッファ間 Credit を利用したフロー制御は基本的にハードウェアカウンタの管理だけで済むため処理が単純であり、またネットワークスイッチのバッファに空きがあれば通信相手ノードのバッファの状態に関わらずパケットの送信が可能なため、スループットの向上が図れる。

4.3.5 軽量通信プロトコルのまとめ

以上をまとめると、信頼性保証のための通信プロトコルの動作は以下のようになる。各パケットは Pkt(送信ノード, 受信ノード, シーケンス番号) で識別されるものとする。

- 送信ノード

  S1 送信ノード $Node(s)$ はパケット $Pkt(s,r,i)$ を受信ノード $Node(r)$ に送信する。$Pkt(s,r,i)$ の再送バッファディレクトリ $Dir(r,i)$ は送信が終っても解放しない。$Pkt(s,r,i)$ のタイムスタンプ $T(r,i)$ をタイムスタンプメモリに保存する。

  S2 $ACK(r,i)$ を受信したら $Dir(r,i), T(r,i)$ を解放する。

  S3 $NACK(r,k)$ を受信したら既に送信した $Pkt(s,r,l)(k \leq l < i)$ を再送する。以後の再送を除く送信を停止する。(新規送信停止)

  S4 新規送信停止中に $ACK(r,k)$ を受信したら $Dir(r,k), T(r,k)$ を解放し、新規送信停止を解除する。

  S5 現在時刻 $Tn -$ タイムスタンプ $T(r,i) >$ タイムアウト時間 To となれば、$Pkt(s,r,i)$ を再送する。

- 受信ノード

  R1 受信ノード $Node(r)$ は、$Pkt(s,r,i)$ を受信し、かつチェックサム正常, 受信側シーケンス番号 $Seq(s) = i$ なら $ACK(r,i)$ を $Node(s)$ に返す。(正常受信)

  R2 受信ノード $Node(r)$ は、$Pkt(s,r,i)$ を受信し、かつチェックサム正常, 受信側シーケンス番号 $Seq(s) > i$ なら受信パケットを破棄し、$ACK(r,i)$ を $Node(s)$ に返す。(重複)

  R3 受信ノード $Node(r)$ は、$Pkt(s,r,i)$ を受信し、かつ受信側シーケンス番号 $Seq(s) < i$ なら受信パケットを破棄し、$NACK(r,Seq(s))$ を $Node(s)$ に返す。(追い越し)

  以後の $Pkt(s,r,j)(i \neq j)$ の受信を拒否する (受信拒否)

  R4 受信ノード $Node(r)$ は、$Pkt(s,r,i)$ を受信し、かつチェックサム異常, 受信側シーケンス番号 $Seq(s) \geq i$ なら受信パケットを破棄し、$NACK(r,i)$ を $Node(s)$ に返す。(チェックサム異常)

  R5 受信拒否中に、再送された $Pkt(s,r,i)$ を受信したら、$ACK(r,i)$ を返し、受信拒否を解除する。

4.3.6 ピギーバック

ここでは、上記の通信プロトコルに対する機能追加として、通信量の削減に効果のあるピギーバックの技術について述べる。

ピギーバック (piggy backing) とは、あるデータパケットに対する確認応答パケットを反対方向のデータパケットに含めて返送する技術であり、TCP [14] においても実現されている。

本通信機構においては、基本的に通信は共有メモリ管理のトランザクションである。そのため、通信は基本的に request,reply の双方向のパケットの組みあわせで行なわれる。

この request,reply パケットはアプリケーションレベルから見ればデータパケットと確認応答パケットに相当するものの、データリンクレベルから見ればどちらも通常のパケットであり、それぞれに対して確認応答が行なわれる。

よって request パケットに対してはデータリンクレベルの確認応答とアプリケーションレベルの確認応答が二重になされることになる。

また、reply パケットに対する確認応答は、アプリケーションレベルの確認応答に対するデータリンクレベルの確認応答であり、これについても冗長である。

このとき、reply パケット内に request パケットに対する確認応答を組み込み、reply パケットに対する確認応

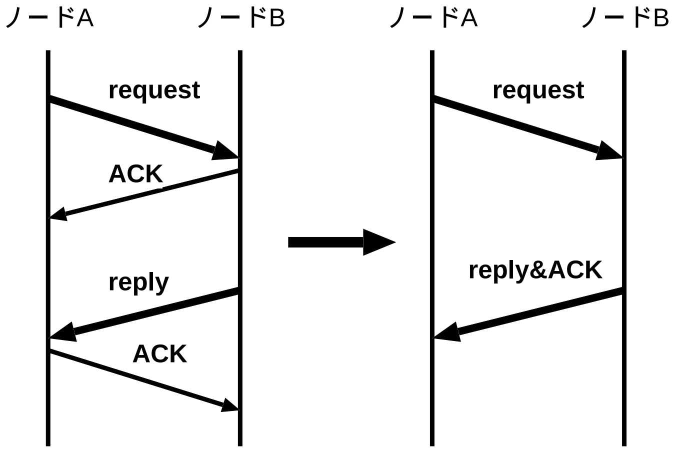

図 6: ピギーバック

答をなくすことで、信頼性を維持しながらネットワークトラフィックの削減が期待できる。

ピギーバックを実装するには前出のプロトコルに若干の変更を加える必要がある。

1. **request** パケットに対する確認応答

   request パケットに対する肯定確認応答 (ACK) の発行を行なわない。ただし追い越しやチェックサムエラーに起因する否定確認応答 (NACK) は request パケットの再送を促すため直ちに発行する。

2. **reply** パケットの再送管理

   reply パケットは再送バッファに登録せず、確認応答も発行しない。reply パケット欠落が発生した場合は request パケットから再送を行なう。

3. 受信 **request** パケットのシーケンス番号の保存

   reply パケット発行に際し request パケットのシーケンス番号が必要になるので、受信側に新たに request パケットのシーケンス番号を保存するメモリが必要になる。

4. **reply** パケットのシーケンス番号

   reply パケット内には、reply パケット自身のシーケンス番号に加え、request パケットの確認応答としてのシーケンス番号も必要になる。

5. **reply** パケットの追い越しに関わる処理

   reply パケット欠落により reply パケットの追い越しが発生する可能性があるので、新たに送信側に確認応答された request パケットのシーケンス番号を保存するメモリが必要になる。また追い越しが発生した場合には確認応答されていない全ての request パケットを再送する。

このピギーバックの実装によって、再送が必要になった場合のコストは大きくなるが、再送が必要ない場合には request,reply パケットに対する確認応答パケットの構成や reply パケットの再送管理にかかわる処理が不要になり、ネットワークトラフィックの削減と共に通信処理に関わるオーバヘッドの削減も期待できる。

4.4 デッドロック対策

ここまでで汎用ネットワークを用いる通信機構における信頼性の問題を解決するネットワーク・インタフェースのプロトコル処理について述べた。しかし汎用ネットワークを用いる通信機構においては、デッドロックが発生する可能性がある。なぜなら、汎用ネットワークにおいては MPP のようにネットワークが多重化されていないからである。そこで上述のプロトコル処理とも関連する本通信機構でのデッドロック対策である、送受信バッファの多重化と再送バッファの利用について述べる。

4.4.1 送受信バッファの多重化

ピギーバックの項で述べたように、本通信機構においては、基本的に通信は共有メモリ管理のトランザクションであり、request,reply の双方向のパケットの組みあわせで行なわれる。

そのため、request パケットを受信したノードではネットワーク・インタフェースからシステムバスに request トランザクションが流され、直ちに reply パケットが構成される。

このとき reply パケットを格納すべき送信バッファに空きエントリがない場合、ネットワーク・インタフェースでは送信バッファに空きができるまで request パケットをブロックしなければならない。このブロック時にデッドロックが発生する可能性がある。

このデッドロックはネットワーク中で reply パケットが request パケットを追い越すことを可能にすれば防げることがある。なぜなら、reply パケットはそのパケットで一つの通信が完了し、送信バッファの空きを必要としないからである。

そのため、専用ネットワークをもつ MPP 等では大容量のバッファを用意したり、バッファや通信路に優先順

位を設け、多重化を行なうことがなされる。

本通信機構においてはメモリ容量の点から、大容量バッファの利用や、送受信バッファの多重化は避けたいところである。

そこで、一つのバッファに対し request,reply 用のキューを用意し、その利用をディレクトリで管理する構成をとる。これにより多少処理は複雑になるが、バッファの容量を倍加させなくても多重化が実現できる。

4.4.2 再送バッファの流用

汎用ネットワークにおいては、上記のように送受信バッファを多重化しても、大抵ネットワークスイッチ内のバッファは多重化されていないため、デッドロック・フリーではない。

そこで、本通信機構ではそれを補う対策として、再送処理の際に利用する再送バッファを流用する。再送バッファによるデッドロック対策は、以下の二点である。

送信前資源確保によるデッドロック予防 送信バッファの容量が小さい場合、確認応答を待つパケットで送信バッファが一杯になることによってデッドロックが発生することがある。そこで、送信時にパケットを再送バッファに確保する。これにより、確認応答パケットの到着を待たずに送信バッファを解放することが可能になる。送信バッファの溢れを防ぐと同時に、送信先のバッファに余裕がある場合には送信バッファの容量に関わらずパケットを送信できるため、スループットの向上も図れる。

資源待避によるデッドロックからの復旧 送信先の問題によりデッドロックが発生した場合、送信先のバッファが一杯でパケットを送信できない状態が続く。そこで、再送時に利用する再送バッファ、フロー制御用の Credit カウンタを利用し、新たにデッドロック検出用タイマを用意することにより、デッドロックの検出を行なう。

ネットワーク・インタフェースは Credit カウンタを常に監視し、送信不可能な状態、すなわち Credit カウンタが最大値になるとデッドロック検出用タイマをスタートさせる。タイムアウトまで Credit カウンタが最大値のままであった場合デッドロックを検出して送信バッファ内の全てのパケットに対して以下の処理を行う。

- 確認応答パケットの場合 :パケットを破棄する
- 確認応答以外のパケットの場合 :パケットを再送バッファに待避させる

破棄もしくは待避されたパケットはネットワーク上で欠落が発生した場合と同様に扱われ、再送処理によって適切に処理される。

ただし、この処理を行なってもデッドロック状態が続く可能性もある。そのような状態が続くと、そのうち再送バッファに空きエントリがなくなってしまう状況が発生する。よって再送バッファへの登録が行なえなくなった時点で、uncorrectable-error として OS の支援を仰ぐ。

これらの対策により、汎用ネットワークを利用しながらもデッドロックの発生を抑止した通信が可能になる。

5 通信機構の実装と予備的評価

本章では、ここまでに示した通信機構の有効性を示すため、通信機構を実装したプロトタイプ・ハードウェアとそのネットワーク・インタフェースの構成を示し、通信処理時間の見積りと考察を行なう。

5.1 プロトタイプ・ハードウェア

Computer Colony は最終的には i-Cartridge をベースにしたハードウェアにより実現することを目指している。しかし、実際にモジュールによる構成を行なう前に、Computer Colony のシステムとしての有効性の評価を行なう環境として、低コストで実現できる分散システムベースのプロトタイプ・ハードウェアを構成する。

5.1.1 プロトタイプ・ハードウェアの構成

本プロトタイプは、低い開発コストで Computer Colony のハードウェアを評価する環境を実現するため、分散システムベースの設計方針をとる。すなわち、従来の分散システムのノードである WS/PC に新たに開発するネットワーク・インタフェースを接続し、高速ネットワークでクラスタを構成するという方針をとる。以下、この方針で製作したプロトタイプ・ハードウェアの構成について述べる。

システムバスを利用できるノード構成 本プロトタイプ・ハードウェアでは、ノードに Sun SPARCstation20 を利用し、そのシステムバスである MBus に新たに

開発するネットワーク・インタフェースを装備する構成をとる。

SPARCstation20の基本性能を表1に示す。システムを構成するノード数は4台とする。SPARCstation20を要素マシンとして採用した理由は以下の通り。

1. 拡張用 **MBus** スロットを利用できる

   Sun SPARCstation20は、マザーボード上に2つのMBusスロットを持つ。1つのMBusスロットには標準でプロセッサを搭載したMBusモジュールが装着されている。もう1つの拡張用MBusスロットにネットワーク・インタフェースを接続することにより、システムバスを利用した通信が行える。

2. **MBus** がキャッシュコヒーレンス制御用信号をサポートしている

   Sun SPARCstation20は、上記の拡張用MBusスロットにプロセッサを搭載したMBusモジュールを装着することにより、SMPに拡張することができる。そのため、MBusではキャッシュコヒーレンス制御のための信号をサポートしている。ネットワーク・インタフェースがこの信号を利用することにより、キャッシュに対しての完全な操作が可能になる。

軽量プロトコルを実現するネットワーク プロトタイプにおけるノード間ネットワークは、プロトコルオーバヘッドが小さい必要がある。本プロトタイプ・ハードウェアでは、高速シリアル通信であるFibre Channelを利用する。Fibre Channelを採用した理由は以下の通り。

1. 低位層のみでの実装が容易である

   低位から高位までプロトコルがレイヤ毎に規格化されているため、必要な部分のみの実装が可能である。

表1: Sun SPARCstation20 の基本性能

CPU	75MHz 動作 SuperSPARC × 1
主記憶	128MByte
1次キャッシュ	16KByte on-chip
2次キャッシュ	1MByte off-chip
キャッシュブロックサイズ	32Byte

2. 部品が比較的手に入りやすい

   Gigabit-Ethernetと互換性のある部品が多いため、比較的安価で手に入りやすい。

プログラマブルデバイスを利用したネットワーク・インタフェース

ネットワーク・インタフェースは、開発コストの低減と、プロトタイプとして様々な動作が可能なように、プログラマブルデバイスを利用して再構成可能なものにする。ただし従来からあるプロトコルプロセッサを装備しソフトウェアを書き換えることで様々な動作を行なうものとは異なり、ハードウェア処理による評価も併せて行なうため、Field Programmable Gate Array(FPGA)を利用した構成とする。

5.2 ネットワーク・インタフェースの構成

ネットワーク・インタフェースでは、前章で説明したリモートメモリのキャッシング, ユーザレベル通信, 軽量通信プロトコル, デッドロック対策を行なう。そのためのネットワーク・インタフェース上の基本的な構成要素は、MBusインタフェース, Fibre Channelインタフェースと各種バッファ、コントローラ、プロトコルプロセッサである。以下では、ネットワーク・インタフェース上の各要素について述べる。

1. **MBus** インタフェース部

   - **MBus** インタフェース・コントローラ

     MBus上の物理アドレスを含むMBusトランザクションを常にスヌープし、MBusに対して適切な処理を行なう。また主記憶タグに対してアクセスを行なう。FPGA内に構成する。

   - 主記憶タグ

     自ノードの主記憶の各キャッシュブロックについて、共有/無効等の情報を保持する。主にリモートメモリキャッシング時のコヒーレンス制御に利用する。ローカルメモリのマッピングを行なうためある程度大容量である必要があり、それほど高速動作を必要としない。そのためSDRAMを利用する。主記憶128MByteに対して2MByteの容量をもつ。

2. プロトコルプロセッサ部

   - プロトコルプロセッサ

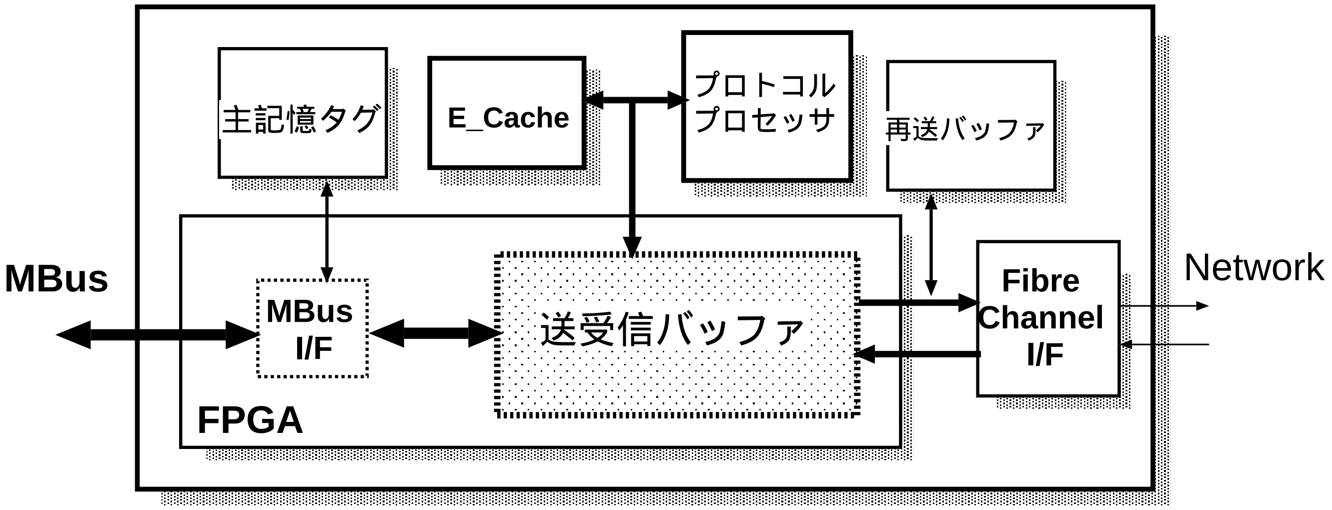

図 7: ネットワーク・インタフェースの構成

主にリモートメモリアクセス時のアドレス変換に必要なテーブル類の管理に用いる。そのほか、ハードウェアで実現困難な機能を実現する。SPARC アーキテクチャに準拠した SPARClite を利用する。このことにより SPARCstation20 上の SuperSPARC と同様のメモリアクセスが行なえる。

- **E_Cache**

  ボード上のプロトコルプロセッサ用の2次キャッシュ。自ノードのメインメモリの内容をキャッシングできる構成にし、プロトコルプロセッサが巨大なメモリ空間を利用できるようにする。キャッシングするデータとしては、アドレス変換のためのページテーブル, 逆引きページテーブルの一部や、送信先ノード特定のためのディレクトリ, パケットのシーケンス番号等である。キャッシュタグ, データアレイ共に高速動作のため SSRAM を用いて構成し、FPGA 内のキャッシュコントローラで管理を行なう。アクセスレイテンシは2サイクル、容量は 1MByte である。

3. 再送バッファ

   送信済パケットを確認応答が到着するまで保存しておくバッファ。高速動作のため E_Cache と同じ SSRAM を用いて構成する。容量は 512KByte である。

4. **Fibre Channel** インタフェース

   主にパラレル/シリアル変換を行なう。市販の Fibre Channel 用 Chip を用いる。

5. 送受信用バッファ

   - **M_Buf** MBus トランザクション用の送受信バッファ。送信側、受信側のバッファをそれぞれ MS_Buf, MR_Buf とする。コマンド処理とデータコピーのオーバラップを可能にするため、それぞれにコマンド用バッファ MSC_Buf, MRC_Buf とデータ用バッファ MSD_Buf, MRD_Buf を別に用意する。
   - **F_Buf** Fibre Channel Frame 用の送受信バッファ。M_Buf と同様にコマンド部とデータ部を分離し、コマンド用バッファFSC_Buf, FRC_Buf とデータ用バッファ FSD_Buf, FRD_Buf を別に用意する。

   コマンド用のバッファは複雑な処理を伴い、また小容量でよいので、FPGA 内のレジスタ・ファイルとして構成する。容量は MSC_Buf, MRC_Buf がそれぞれ 8Byte × 4 エントリ、FSC_Buf, FRC_Buf がそれぞれ 40Byte × 2 エントリである。データ用のバッファは FPGA 内のデュアルポートメモリで構成する。容量は MSD_Buf, MRD_Buf がそれぞれ 32Byte × 4 エントリ、FSD_Buf, FRD_Buf がそれぞれ 512Byte × 2 エントリである。

6. その他のメモリ

信頼性保証処理のため、ネットワーク・インタフェース上ではさらに以下のメモリをFPGA内に用意する。

- 再送バッファのディレクトリ
- タイムスタンプメモリ
- Credit 管理用レジスタ

5.3 通信処理時間の見積り

本節では、ネットワーク・インタフェース上の通信処理時間の見積りを行なう。その目的は、ネットワーク・インタフェースが細粒度並列処理に必要な通信性能を実現できるかを確認するためである。基本性能測定のため、リモートメモリリードについて見積りを行なった。

5.3.1 トランザクションとパケット

通信時にネットワーク・インタフェース内で処理されるのは以下のものである。なお、Fibre Channel ではパケットのことをフレームと呼ぶが、本稿ではパケットに統一する。

- read_request トランザクション :8Byte のコマンド部からなる MBus トランザクション
- read_request パケット :36Byte のヘッダ、フッタと 8Byte の MBus トランザクション、4Byte の仮想空間 ID の合計 48Byte のコマンド部からなる Fibre Channel パケット
- read_reply トランザクション :8Byte のコマンド部と 32Byte のデータ部からなる MBus トランザクション
- read_reply パケット :read_request パケットと同様の 48Byte のコマンド部と、32Byte のデータ部からなる Fibre Channel パケット

これらの関係を図8に示す。図のように、MBus のトランザクションを Fibre Channel のパケット内にカプセル化する。MBus トランザクションにおいてはコマンド部の中に物理アドレスが存在するが、パケット内では仮想アドレスに変換したものを格納している。

5.3.2 通信処理のフェーズ

ネットワーク・インタフェース上の通信処理は、送受信バッファによって図9の6つのフェーズに分かれる。

このうちフェーズ2, フェーズ5については、ハードウェアで行なう処理の他に、プロトコルプロセッサ上のソフトウェアで行なう処理が可能である。これらの処理の切り分けをどのように行なうかは重要な問題であるが、現在プロトコルプロセッサ部が未実装のため、フェーズ2,5については、通信に最低限必要な処理を行なうハードウェアが FPGA 内に存在することを仮定し、見積りを行なう。すなわち、フェーズ2, フェーズ5ではプロトコルプロセッサは動作せず、FPGA 内のハードウェアが E_Cache 上のテーブルを参照するだけで通信に必要な処理を行なう。

フェーズ2, フェーズ5以外のフェーズに関しては、実装に基づいた処理時間の見積りを行なう。

5.3.3 各フェーズ処理時間の見積り

フェーズ1 フェーズ1では、MBus 上にトランザクションが送出されてから MS_Buf に格納するまでの処理を行なう。この時の処理は通信の必要性の関知と、トランザクションの格納である。

1. 通信の必要性の関知

   トランザクションの物理アドレスに対し通信が必要かどうか判断する。主記憶タグを物理アドレスで引き、その出力によって判断する。全ての処理には6サイクルかかる。

2. トランザクションの格納

   通信の必要性の関知に並行してトランザクションの格納を行なう。

3. フェーズの終了通信の必要性の関知, トランザクションの格納の双方が完了し、通信が必要であればエンキューを行なってフェーズを終了する。

read_request の場合、必要なサイクル数は7サイクルである。

read_reply の場合、バスに read_request のアドレスが送出された後8サイクル後にメモリからデータが送出されるため [15]、必要なサイクル数は12サイクルである。

フェーズ2 フェーズ2では、通信の必要なバストランザクションが MS_Buf に格納されてから、パケットを構成して FS_Buf に格納するまでの処理を行なう。この時の処理は、パケットヘッダの構成のための処理と MSD_Buf → FSD_Buf へのデータコピーである。

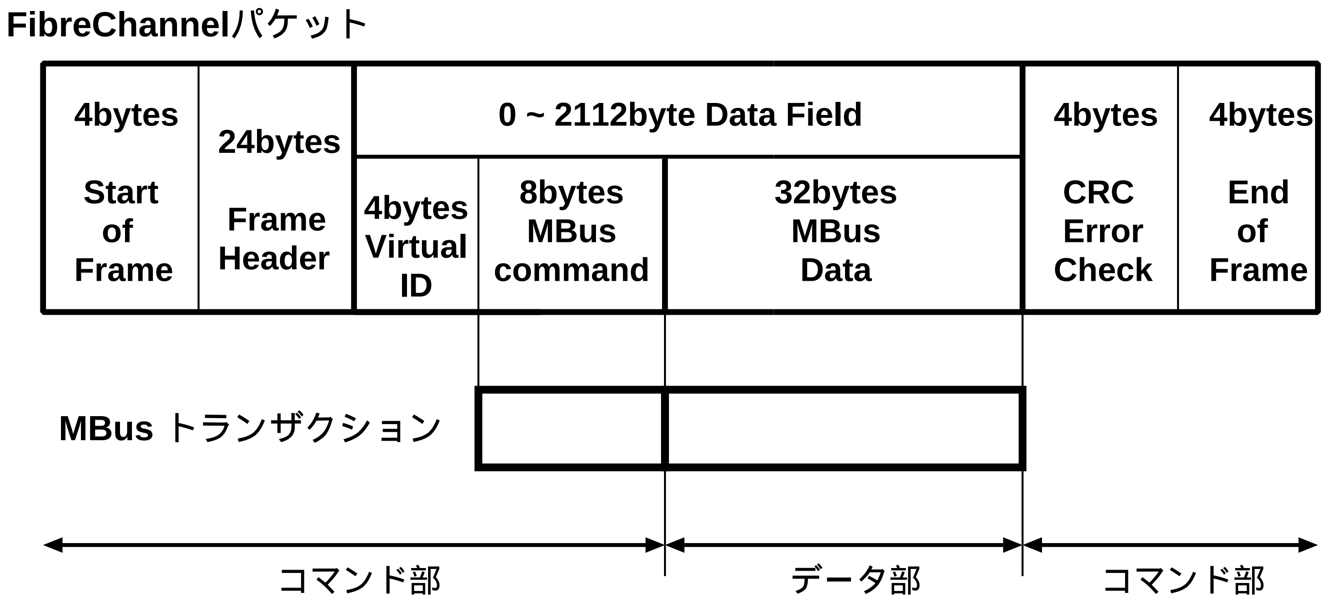

図 8: トランザクションとパケット

5.3.2 で述べたように、本フェーズではパケットヘッダの構成のため FPGA 内のハードウェアが E_Cache 上のテーブルを参照することにより処理を行なう。

1. パケットヘッダ構成
   (a) 物理アドレス → 仮想アドレス変換

   システムバス上の物理アドレスをネットワークアドレスの構成のために仮想アドレスに変換する。MSC_Buf[PA] の物理アドレスをキーとして E_Cache 上の RTLB を引き、出力した仮想アドレスを FSC_Buf[VA] に、仮想空間 ID を FSC_Buf[V_ID] に格納する。

   (b) 送信先ノード番号の読み出し

   仮想空間 ID から送信先のノード ID(D_ID) を得る。FSC_Buf[V_ID] をキーとして E_Cache 上のノード情報ディレクトリを引き、出力を FSC_Buf[D_ID] に格納する。

   (c) シーケンス番号の読み出し

   D_ID からパケットに与えるシーケンス番号 (seq_ID) を得る。FSC_Buf[D_ID] をキーとして送信側シーケンス番号を引き、FSC_Buf[seq_ID] に格納する。

2. データコピー

   MSD_Buf → FSD_Buf へのデータコピー

必要なサイクル数はパケット構成処理に 13 サイクル、N word のデータコピーに $N+2$ サイクルである。

パケット構成処理とデータコピーは並行処理が可能である。なお、1 word = 16 bit である。

必要なサイクル数は、read_request で 14 サイクル、read_reply で 19 サイクルである。なお、サイクル数には E_Cache のキャッシュヒット判定も含まれる。

フェーズ 3　フェーズ 3 では、送信バッファにパケットが用意されてからネットワークにパケットが送出されるまでの処理を行なう。この時の処理は以下のものである。

1. フラグチェック

   通信可能状態かどうか調べるためバッファ間クレジットをチェックする。データパケットであればさらに再送バッファへの登録が可能であるか調べるため、再送バッファディレクトリのエントリがチェックされる。またタイムスタンプの設定を行なう。

2. パケット送信

   通信可能状態であれば、ネットワークにパケットが送信され、並行して再送バッファへの登録を行なう。

必要なサイクル数は read_request パケットで 27 サイクル、read_reply パケットで 43 サイクルである。

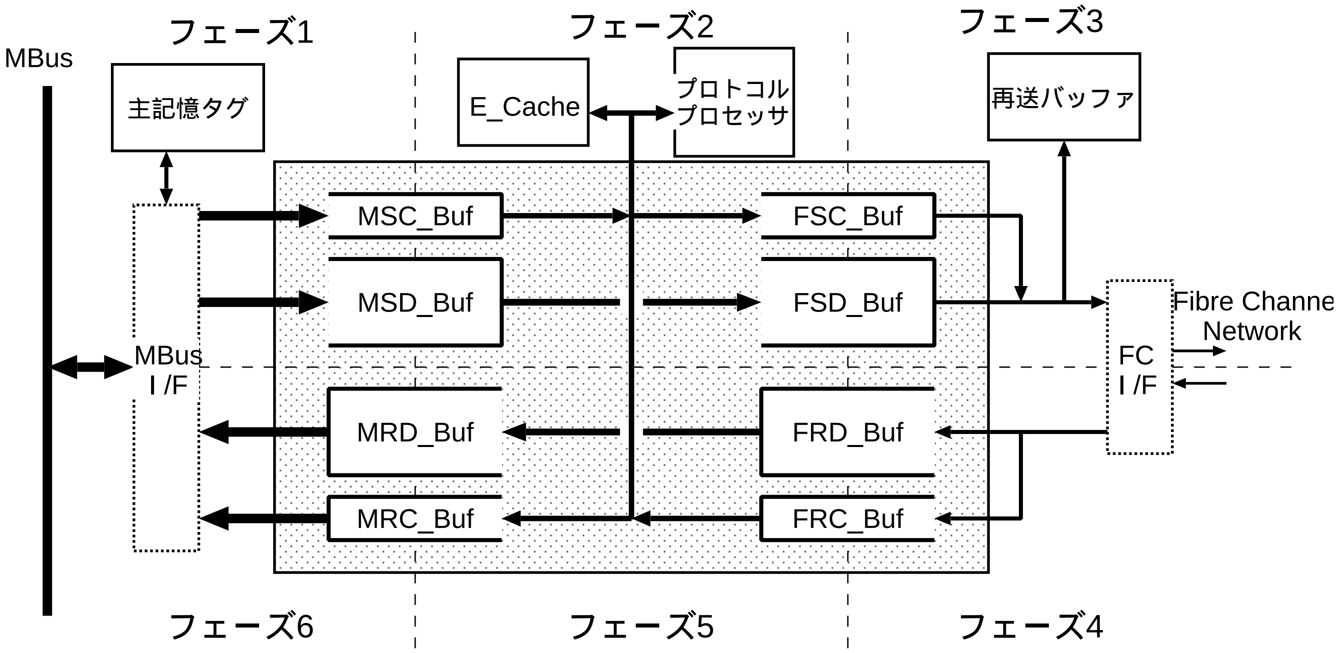

図 9: 通信処理の各フェーズ

フェーズ 4　フェーズ4では基本的にネットワークから送信されるパケットを FR_Buf に格納するだけである。必要なサイクル数は read_request パケットで 26 サイクル、read_reply パケットで 42 サイクルである。

フェーズ 5　フェーズ5では、受信バッファFR_Buf にパケットが到着してからバストランザクションを MR_Buf に格納するまでの処理を行なう。受信バッファにパケットが到着すると、以下のような処理を行なう。

1. パケットの種類チェック

   パケットの種類を表すヘッダ内の R_CTL 部と S_ID チェックによってパケットの送信元と種類を調べる。

2. パケットの有効性チェック

   seq_ID チェックによってパケットの有効性を調べる。

3. 確認応答処理

   受信パケットが確認応答パケットの場合、確認応答処理として再送バッファディレクトリの無効化, タイムスタンプの無効化を行なう。

4. 仮想アドレス → 物理アドレス変換

   正しいデータパケットを受信した場合には、仮想アドレス → 物理アドレス変換が行なわれる。この時の処理は、FPGA 内の FRC_Buf の仮想アドレスをキーとして E_Cache 上の TLB を引き、出力を MRC_Buf[PA] に格納することによって行なう。

5. データコピー

   トランザクションのデータ部を FRD_Buf から MRD_Buf へコピーする。

必要なサイクル数は、read_request パケットで 12 サイクル、read_reply パケットで 22 サイクルである。

フェーズ 6　フェーズ6で行なう処理は、バスの獲得と MR_Buf から MBus へのトランザクションの送出である。

必要なサイクル数は、read_request で 5 サイクルである。また、バスの獲得が必要ない read_reply では 5 サイクルである。

単方向レイテンシ　ここまでの各フェーズの処理時間の見積りより、単方向のレイテンシを求めると、表 2のようになる。

5.3.4 レイテンシについての考察

本節では、通信性能に関するネットワーク・インタフェースの有効性を示すため、見積りを行なったレイテ

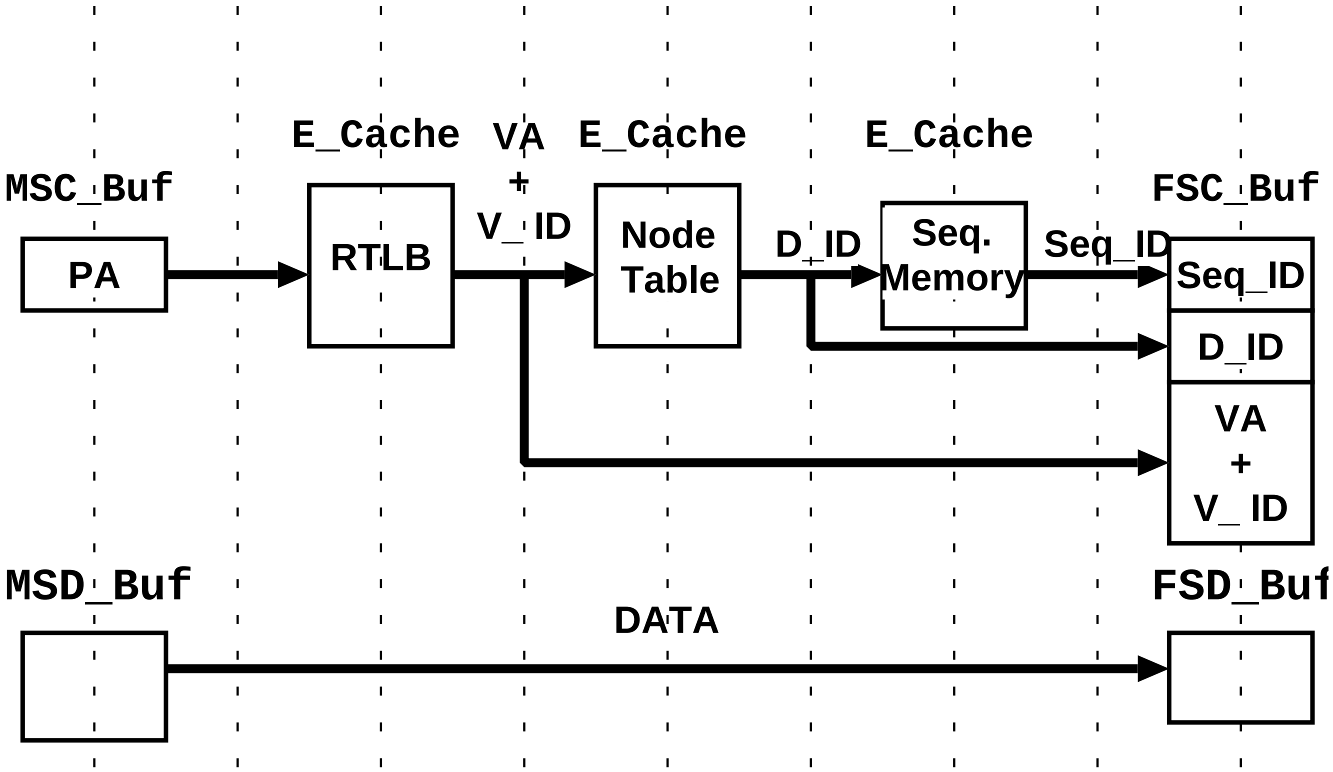

図 10: フェーズ 2 における処理

表 2: 単方向レイテンシ

トランザクション	1	2	3	4	5	6	合計
read_request	7	14	27	26	12	5	91 サイクル
read_reply	12	19	43	42	22	5	143 サイクル

ンシについての考察を行なう。

表2より、ネットワーク・インタフェースにおけるリモートメモリへの read アクセスのラウンド・トリップ・レイテンシは 234 サイクルである。本ネットワーク・インタフェースは基本的に 50MHz で動作するため、このレイテンシは 4680ns であり、細粒度並列処理に必要な μs 単位のレイテンシを実現するものである。

ただし、このレイテンシはアドレス変換等をハードウェアで行なったと仮定した、ある程度理想的な場合のものである。このような場合に高速処理が可能である最も大きな要因は、フェーズ 2 における物理アドレスからネットワークアドレスへの変換と、フェーズ 5 におけるネットワークアドレスから物理アドレスへの変換を行なう際に、必要な情報が E_Cache 上にキャッシングされていることである。

通信時に参照されるこれらの情報は、リモートメモリアクセスを行なうアドレスに対し、ページ単位での対応づけを行なうものである。そのため、同一ページ内のキャッシュブロックに対するメモリアクセスが連続して発生するような場合には、直前のアクセスで参照した情報を後続のメモリアクセスがそのまま利用することができる。

共有メモリにおける通信はメモリアクセスであるため、通信処理自体も参照の局所性を内在する。本通信機構では、ネットワーク・インタフェース上に比較的大容量の E_Cache を装備することによって、通信処理に関し

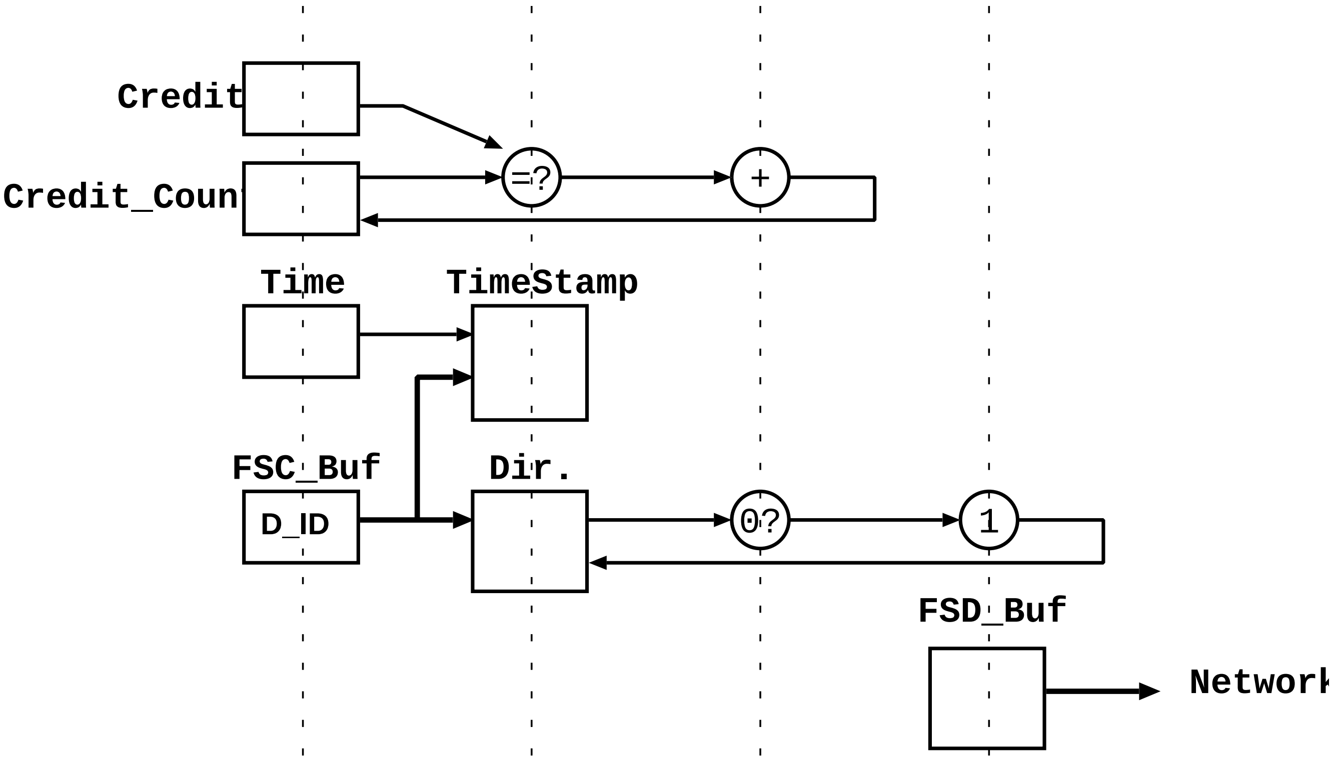

図 11: フェーズ 3 における処理

て積極的に局所性を利用することが可能になり、通信処理の高速化に有効であると考えられる。

6　おわりに

本稿では、LAN 程度の規模の分散システムにおいて比較的細粒度の並列処理を実用レベルにまで高めることを目指すネットワーク分散型並列処理システムを実現する通信機構について述べた。

本通信機構では、分散環境における通信の問題である、レイテンシ, スループットの不足、通信の信頼性、優先順位の不足の問題を解決し、分散システムにおいて MPP に比肩する性能の実現を目指す。そのため、キャッシングによる通信最適化のサポート, ユーザレベル通信の実現, 軽量プロトコルによる通信の信頼性保証, デッドロック対策、という 4 つのアプローチをとった。

本通信機構では、システムバスに接続したネットワーク・インタフェース上にメインメモリのキャッシュタグを備え、コヒーレンス制御を行なう。このことによりメインメモリをリモートメモリを大容量キャッシュに利用でき、通信の絶対量を減らす最適化のサポートを可能にする。

またネットワーク・インタフェースが物理アドレスで通信を起動することにより、通信資源の保護を実現し従来の分散システムで必要であったシステムコールのオーバヘッドをなくすことができる。さらに通信時に必要なアドレス変換を高速に行なうため、ネットワーク・インタフェース上に主記憶のキャッシングを行なえる機構を提供した。

従来プロトコルプロセッサで行なわれていた信頼性保証処理であるパケットの順序管理, 送信パケットの管理, 再送処理, フロー制御に対しハードウェアサポートを行なう。このことにより、プロトコル処理全体に関わるオーバヘッドを削減することが可能である。

さらにネットワーク・インタフェース上の送受信バッファの多重化と信頼性保証処理に利用する再送バッファを流用することによって、デッドロックの防止を行なった。

本通信機構をプロトタイプ・ハードウェアのネットワーク・インタフェースへ一部実装を行なった。現在ネットワーク・インタフェース上のプロトコルプロセッサ部

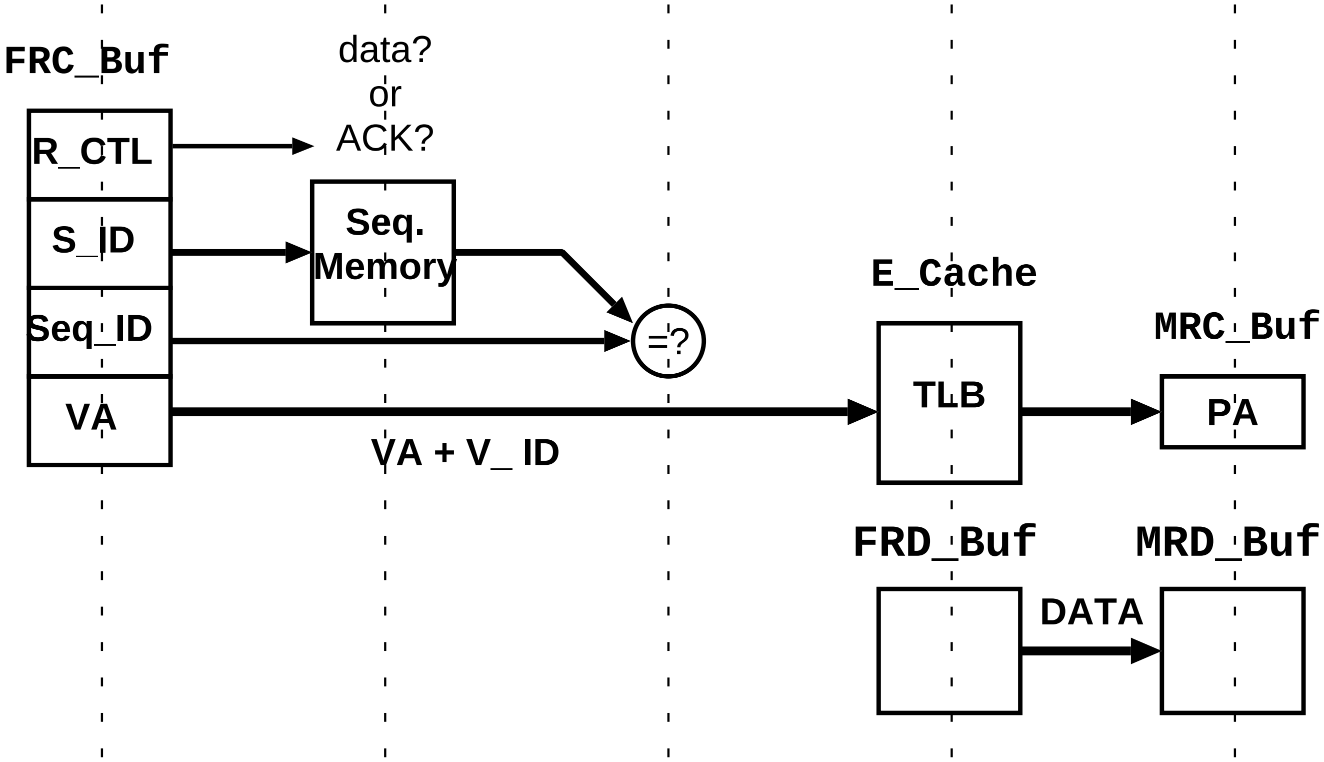

図 12: フェーズ 5 における処理

が未実装であるため、理想的なハードウェアの存在を仮定した上での見積りを行なった。その結果、キャッシュミスヒット時のリモートメモリアクセスが 234 サイクル程度で可能であることを確認した。

本通信機構では、通信処理に必要な情報をネットワーク・インタフェース上にキャッシングを行なう。そのため、参照の局所性を積極的に利用することが可能になり、細粒度並列処理に必要なμs 単位のレイテンシを実現するために有効であると考えられる。

謝辞

本研究の機会を与えて頂いた、本研究室の富田眞治教授に深甚な謝意を表します。

また、本研究に関して適宜御指導、御鞭撻を賜った森眞一郎助教授、五島正裕助手に深く感謝致します。

さらに，共同研究者である Damien Le Moal 氏、増田峰義氏、朝川 達氏をはじめとして，日頃様々な角度から助力して下さった京都大学大学院情報学研究科通信情報システム専攻富田研究室の諸兄に心より感謝致します。

参考文献

[1] X3T11 Task Group of ANSI: Fibre Channel Physical and Signaling Interface (FC-PH), Rev. 4.3 June 1,(1994).

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and Wen-King Su. “Myrinet - A Gigabit-per-Second Local-Area Network” IEEE MICRO, Vol. 15, No.1,pp.29-36,February 1995.

[3] B. Chun, A. Mainwaring, and D. Culler, “Virtual Network Transport Protocols for Myrinet,” IEEE MICRO, pp.53-63,Jan 1998.

[4] S. Pakin, M. Lauria, and A. Chien, “High Performance Messaging on Workstations: Illinois Fast Messages(FM) for Myrinet,”*Proc. Supercomputing’95*,ACM Press, New York, 1995.

[5] C. Dubnicki et al,”Myrinet Communication,” IEEE MICRO, pp.50-52,Jan 1998.

[6] 手塚 宏史他 :ワークステーションクラスタ用通信ライブラリPMの設計と実装, 並列処理シンポジウムJSPP’96論文集, pp. 41-48.(1996).

[7] 住元 真司他 Gigabit Ethernetを用いた高速通信ライブラリの設計と評価, 並列処理シンポジウムJSPP’99論文集, pp.63-70.(1999).

[8] Mark D. Hill, James R. Larus, and David A. Wood, ”Tempest: A Substrate for Portable Parallel Programs,” Proc. of Joint Symp. on Parallel Processing, pp.123–128, May 1995.

[9] Wolf-Dietrich Weber, et al., “ The Synfinity Interconnect Architecture : A Cost-Effective Infrastructure for High - Performance Servers, ” (URL http://www.fjst.com/products/synfinitynuma/abstract/)

[10] Marco Fillo and Richard B. Gillet, “ Architecture and Implementation of MEMORY CHANNEL2 ”, *DIGITAL Technical Journal*(on line), 28 Aug. 1997. (URL http://www.digital.com/info/DTJP03/DTJP03HM.HTM)

[11] Kai Li, Paul Hudak :Memory Coherence in Shared Virtual Memory Systems, In the ACM Transactions on Computer Systems, Vol7, No.4, November 1989, pp321-359

[12] 佐藤 充 他: 超並列マシンJUMP-1のための分散共有メモリ管理プロセッサ, 並列処理シンポジウムJSPP’97論文集, pp. 265-272.(1997).

[13] 増田 峰義: 分散OS Coloniaにおける共有メモリを利用した大域的ネーム・サービス, 特別研究報告書, 京都大学工学部情報工学科(1999).

[14] Douglas Comer: TCP/IPによるネットワーク構築 Vol.1 – 原理, プロトコル, アーキテクチャ(1995).

[15] Ben Catanzaro: SPARCマルチプロセッサアーキテクチャ(1995).

[16] 伊達 新哉 他: コンピュータ・コロニーを実現する高速通信機構, 信学技報 99-CPSY-52(SWoPP 下関’99), pp.41-48.